Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dominik Grah

Procesno rudarjenje s programom ProM

diplomsko delo

univerzitetni študij računalništva in informatike

Mentor: izr. prof. dr. Marko Bajec

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [**?**]

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Procesno rudarjenje s programom ProM

Tematika naloge: Številni informacijski sistemi beležijo, kdaj se posamezna funkcionalnost sistema uporablja, kdo jo uporablja ipd. Dnevniki, v katere se ti podatki zapisujejo, se izkažejo zelo uporabni za različne analize, med drugim za analizo procesnih informacij oziroma za ugotavljanje, kakšni procesi so se izvajali v okolju, ki ga informacijski sistem podpira. Za analizo obstajajo številna orodja.

V okviru diplomske naloge preučite področje procesnega rudarjenje. Na praktičnem primeru prikažite uporabo odprto-kodnega orodja ProM.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dominik Grah, z vpisno številko **63060055**, sem avtor diplomskega dela z naslovom:

Procesno rudarjenje s programom ProM

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Bajeca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30. maja 2015 Podpis avtorja:

Najlepše se zahvaljujem prof. dr. Marko Bajec za pomoč pri izdelavi diplomske naloge. Prav tako se pa zahvaljujem staršem za podporo in možnost študija.

Kazalo

Povzetek

Osnovni namen diplomskega dela je prikazati uporabo procesnega rudarjenja na realnem primeru. Prav tako pa raziskati in podati prednosti ter slabosti procesnega rudarjenja. V teoretičnem delu so podrobneje predstavljeni: namen uporabe procesnega rudarjenje, njegove možnosti za uporabo, različni algoritmi, prav tako pa nekaj možnih procesnih modelov za predstavitev procesa. V zadnjem delu teoretičnega dela pa je predstavljeno orodje ProM in ostala možna orodja, prav tako pa primerjava med njimi. V praktičnem delu pa je predstavljena uporaba prikazanih tehnik na realnem primeru iz okolja. Kot rezultat analize s programom ProM dobimo različne modele, mreže in grafe, s pomočjo katerih lahko potem analiziramo proces.

Ključne besede

Procesno rudarjenje, procesni model, petrijeve mreže, c-mreže, dnevnik dogodkov, alfa algoritem, hevristično rudarjenje, inductive miner, fuzzy miner, token replay, organizacijski vidik, časovni vidik.

Abstract

The primary purpose of this diploma is to demonstrate the use of process mining on a real life case and also to explore advantages and disadvantages of process mining. The theoretical part presents in detail the purposes and reasons of applying process mining to organizations, different algorithms for process mining, and some possible process models for representation of processes. The last chapter in theoretical part presents the ProM tool, which is used for process mining, and also other popular tools. The last part is practical part, which shows the usage of process mining techniques on a real life example. As a result of analysis with program ProM, we get different models, networks and graphs, which we can then use to analyze the process.

Key words

Process mining, process model, Petri nets, c-nets, event logs, alpha algorithm, heuristic mining, inductive miner, fuzzy miner, token replay, organizational mining, time perspective.

Poglavje 1

Uvod

V današnjem času imajo informacijski sistemi vse večjo veljavo, prav tako pa postajo vse bolj prepleteni s procesi, ki jih podpirajo. Zaradi nadzora, varnosti, možnosti izboljšave in številnih drugih razlogov, beležijo vse več dogodkov v njih. Pri tem pa nastane problem, saj podjetja ne znajo izvleči uporabnih informacij z ogromnih količin zabeleženih dogodkov. Ravno pri tem pa stopi v veljavo procesno rudarjenje. S pomočjo različnih tehnik poizkuša dobiti čim več uporabnih podatkov. Namen vsega tega pa je povečati učinkovitost in uspešnost procesov, prav tako pa odkriti pomanjkljivosti in nepravilnosti v njih. Z izboljšanjem procesov pa se zmanjšajo stroški, poveča učinkovitost poslovanja, število strank in posledično veča dobiček. Procesno rudarjenje je relativno mlada tehnika, katera se je začela uporabljati ne tako dolgo nazaj. Temelji na procesnem modelu in podatkovnem rudarjenju, vendar pa je dosti več kot samo združitev obstoječih pristopov, kajti pri podatkovnem rudarjenju namenimo preveč pozornosti samim podatkom. Posledica tega pa je, da ne moremo dobiti natančnega pregleda nad celoto procesa. Ostala področja, kot je poslovna inteligenca, pa se fokusira na preprosta namizja in poročila, pozablja pa na podrobnejši pregled procesov. Področje upravljanja poslovnih procesov pa se preveč posveča idealiziranemu pogledu na procese, pozablja pa na dejansko stanje procesov. Zaradi omenjenih razlogov je procesno rudarjenje koristna pridobitev za vsa omenjena področja.

Trenutno najbolj uporabljeno orodje za uporabo tehnik procesnega rudarjenja je ProM. ProM je brezplačni odprtokodni program, ki vsebuje vse pogostejše tehnike procesnega rudarjenja. Prednost programa ProM pa je tudi v tem, da je možno razširiti dodatne možnosti z uporabo različnih vtičnikov. Vtičnike pa lahko kdorkoli razširi in doda v sam program pri tem pa ni potrebno ponovno prevajanje programa. Dodamo samo zapis v nekaj .ini datotek.

Diplomska naloga je sestavljena iz dveh delov. V prvem delu je teoretični opis procesnega rudarjenja, tehnik rudarjenja ter možnih orodij, v drugem delu pa je uporaba tehnik procesnega rudarjenja na primeru iz realnega okolja s programom ProM.

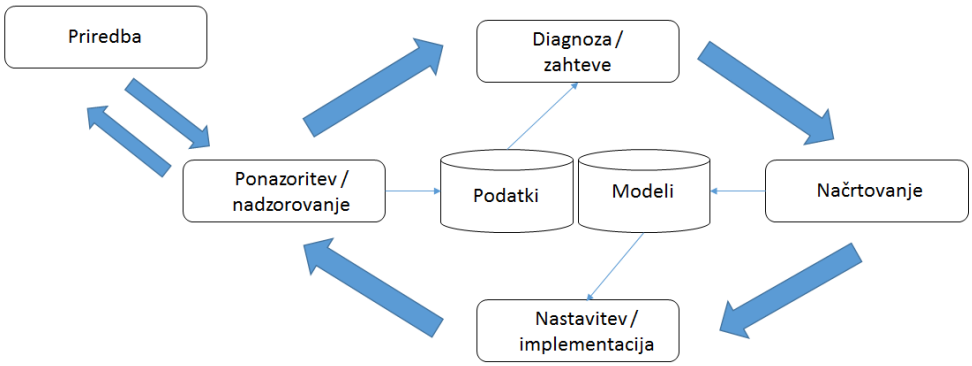
Poglavje 2

Procesno rudarjenje

2.1 Definicija in uvrstitev procesnega rudarjenja

Procesno rudarjenje je skupek tehnik upravljanja procesov, katere omogočajo analizo poslovnih procesov na osnovi dnevnika dogodkov, pri čemer pa kot proces označimo skupek aktivnosti, ki prejmejo enega ali več vrst vhodov in ustvarijo izhod, ki je pomemben za stranko. Poslovni proces ima cilj, nanj pa delujejo dogodki iz zunanjega sveta ali iz drugih procesov. [1] Večina procesov, ki potekajo v podjetjih je podprtih s strani informacijskega sistema. Prav tako je v navadi, da informacijski sistem beleži vsak dogodek, ki se zgodi v njem, ter ga shrani v dnevnik dogodkov. Glavna ideja procesnega rudarjenja je izvleči čim več znanja in informacij iz dnevnikov dogodkov zapisanih s strani informacijskega sistema. Glavni cilj in s tem tudi namen rudarjenja procesov je izboljšanje poslovanja s tem, da poda orodja in tehnike za odkrivanje procesov, podatkov, organizacijske in družbene strukture, ter njihovega nadzora. To pa naredi na takšen način, da podatke, ki so potrebni za vse to, pridobi iz dnevnikov dogodkov.[2]

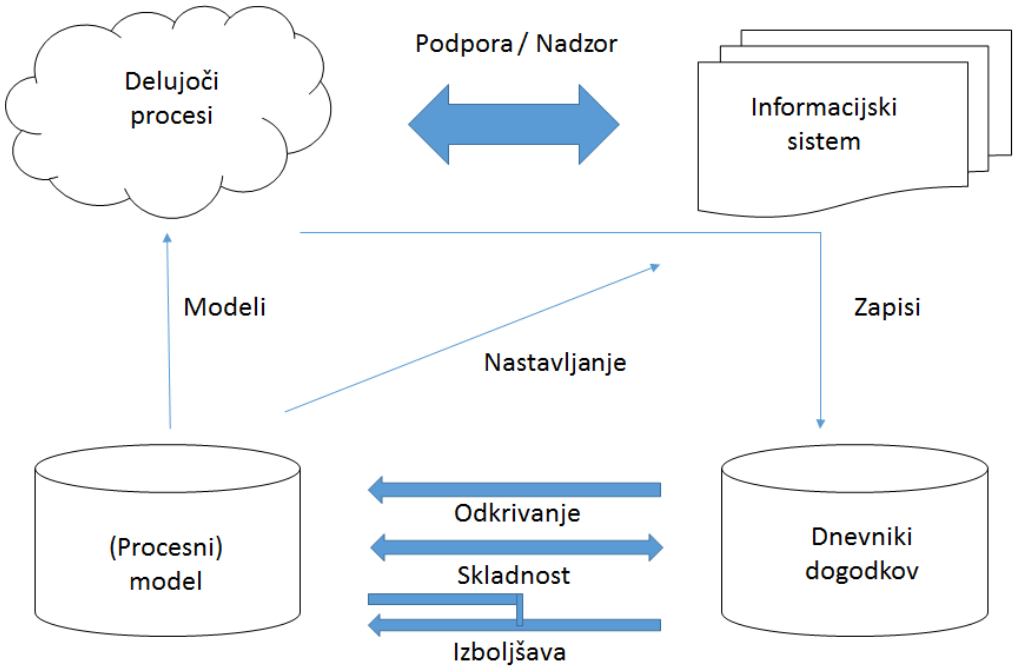
V današnjem času se z analizo in z izboljšanjem procesov ukvarjajo že nekatera področja. To so upravljanje poslovnih procesov (BPM), poslovna inteligenca (BI), upravljanje delovnih tokov (WFM) ter še nekatera. Kar je



Slika 2.1: BPM življenski cikel [3]

pravzaprav razumljivo, saj imajo velik potencial pri povečanju produktivnosti in zmanjšanju stroškov. Da bi lažje uvrstili in razumeli glavno razliko med rudarjenjem procesov ter ostalimi področji moramo najprej pogledati BPM življenjski cikel(Slika 2.1).

Življenjski cikel je sestavljen iz različnih faz upravljanja določenega poslovnega procesa. V fazi načrtovanja se sestavi procesni model in opravi vse stvari, ki jih bomo pozneje potrebovali za implementacijo oz. se ponovno načrtuje, če je proces že bil načrtovan. Ta se nato v fazi nastavitev/implementacije pretvori v delujoči sistem. Če je model že v izvrševanju ter WFM ali BPM sistem že teče, je ta faza zelo kratka. V primeru, da je model še samo informativen, ter ga je potrebno dejansko sprogramirati se ta faza lahko zelo zavleče. Potem sledi faza ponazoritve/ nadzorovanja. Tukaj vsi procesi tečejo, medtem ko jih nadzira upravitelj z namenom, da bi ugotovil, če so potrebne spremembe. Nekatere izmed teh sprememb se upoštevajo v naslednji fazi, ki je faza priredbe. V tej fazi se ne ustvari ali spreminja noben del programske opreme ali kakorkoli ponovno načrtuje procese. Spremeni se le možne nastavitve, že vgrajene v programsko opremo. V fazi diagnoza/zahteve se programska oprema ovrednoti ter iz razlogov, kot so slaba uspešnost ali zahteva za nove možnosti, ponovno sproži fazo načrtovanja. Pri prikazanem modelu igra procesni model glavno vlogo v



Slika 2.2: Procesno rudarjenje kot povezava med modelom in podatki [4]

fazi načrtovanja in nastavitev/implementacije. Podatki pa igrajo pomembno vlogo v fazi ponazoritve/nadzorovanja in diagnoza/zahteve.

Ravno zaradi tega nastane razlika med dejansko pridobljenimi podatki in procesnim modelom, kajti dejanskih podatkov se ne uporabi pri načrtovanju procesnega modela.

Zato je pri procesnem rudarjenju glavna težnja proti temu, da postavimo model že na osnovi zapisanih dnevnikov dogodkov ter s tem to pomanjkljivost odpravimo.

Kot lahko vidimo na sliki 2.2 procesno rudarjenje vzpostavi povezavo med dejanskim procesom in njegovimi podatki ter modelom procesa. Današnji informacijski sistemi zabeležijo ogromno število dogodkov. Klasični WFM sistemi, BPM sistemi, ERP sistemi, CRM sistemi, PDM sistemi, ... podajajo podroben pregled kaj se z aktivnostmi in sistemom dogaja. Na 2.2 se ti zabeleženi dogodki vidijo v dnevniku dogodkov. Vendar pa moramo

biti pozorni na dejstvo, da večina informacijskih sistemov shranjuje takšne informacije v nestrukturirani obliki, npr. dnevniki dogodkov so razmetani po različnih tabelah ali pa jih je potrebno odcepiti od podsistema za izmenjavo sporočil. Zato je vsako črpanje podatkov pomemben del pri procesnem rudarjenju. [3]

2.2 Uporaba procesnega rudarjenja

Na podlagi zbranih dnevnikov dogodkov lahko izvedemo tri različne načine procesnega rudarjenja:

- Odkrivanje(discovery), je tehnika izgradnja procesnega modela brez predhodno pridobljenih informacij o samem sistemu. S to tehniko preprosto vzamemo dnevnik dogodkov in ta poda rezultat v obliki procesnega modela. Primer take tehnike je alfa algoritem. Ta vzame dnevnik dogodkov in poda petri mrežo, katera razloži dogajanje v samem modelu.

- Skladnost(conformance), pri tej tehniki obstoječi model primerjamo z dnevnikom dogodkov enakega procesa. Ta tehnika se na primer lahko uporablja pri preverjanju, če realnost zadošča modelu. Se pravi, če se v realnosti zgodijo taki dogodki in po takšnem vrstnem redu, ko je postavljeno v samem procesnem modelu. Prav tako se lahko preveri princip »štirih oči«, se pravi, da se neka aktivnost ne opravi le z strani ene in iste osebe, kajti s tem ko jih pregleda več oseb, se zmanjša verjetnost za napako. Iz vsega tega sledi, da se ta tehnika uporablja za odkrivanje, iskanje in pojasnjevanje odstopanj ter resnost odstopanj od zastavljenega modela.

- Izboljšava(enhancment), zadnja tehnika pa se uporablja za razširitev ali izboljšavo obstoječega procesnega modela z uporabo dejanskih informacij, pridobljenih na procesih.

[5]

Proces pa lahko pogledamo iz več različnih perspektiv, ter na podlagi perspektive pridemo do različnih zaključkov. Možne perspektive pa so naslednje:

- Perspektiva nadzora toka se fokusira na potek procesov. Se pravi, želi najti najboljšo karakterizacijo vseh možnih poti v procesu. Obstajajo različni načini prikaza kot so Petri mreže, Workflow mreže, BPMN, YAWL, ... V tem primeru nas najbolj zanima izgradnja dejanskega procesnega modela.

- Organizacijska perspektiva se fokusira na informacije o resursih, skritih v log datotekah. Zanima nas kje so resursi(ljudje, sistemi, vloge in oddelki) udeleženi in kako so povezani. Cilj je prikazati sestavo organizacije, tako da klasificira ljudi in vloge ali pa izdelamo socialno mrežo resursov.

- Perspektiva na strani primera pa se osredotoča na lastnosti primerov. Seveda se lahko primer karakterizira s strani poti v procesu ali tistih, ki delajo na primeru ali pa s strani pripadajočih podatkovnih elementov.

- Časovna perspektiva se ukvarja z časom in pogostostjo pojavljanja dogodkov. Če primer vsebuje časovni žig, se lahko z procesnim rudarjenjem ugotovijo ozka grla, nadzorujejo zasedenost in uporabnost resursov, napove preostali čas procesiranja primerov.

Potrebno je pa dodati, da se lahko različne perspektive med sebpj prekrivajo, kljub temu pa podajo dober pregled nad tem, kaj želi procesno rudarjenje doseči.

Obstajata dva načina kako se lahko izvede procesno rudarjenje. Prvi način in tudi najbolj pogost je, da se analiza naredi off-line, torej se opravi pregled po poteku procesa. Njen namen je izboljšanje procesa ali vzpostavitev boljšega razumevanja procesa, vendar pa je možno vse več tehnik uporabiti tudi v drugem načinu analize, se pravi on-line, med samim potekom procesa.

To je na primer podajanje časa, ki ga en primer v procesu rabi za dokončanje na podlagi prejšnjih podatkov.[3]

Poglavje 3

Procesni modeli ter njihova analiza

Procesni modeli igrajo zelo pomembno vlogo pri vseh večjih načrtovanjih, izboljševanju oz. optimizaciji informacijskih sistemov. Obstajajo organizacije, ki uporabljajo le neformalne procesne modele, in sicer za diskusijo ali dokumentacijo procedur. Vendar pa vse ozaveščene organizacije glede pomena procesnih modelov, jih ne uporabljajo le za to, saj so procesni modeli sestavni del analize procesov in izboljšajo delovanje le teh. Dandanes se večina procesnih modelov naredi "na roko", brez kakšnih strogih analiz ali upoštevanja že obstoječih podatkov, zato je procesno rudarjenje lahko še kako pomembno pri načrtovanju, saj upošteva že zbrane podatke, prav tako pa opravi obsežne analize, preden naredi model. Dober procesni model pa ima še posebno pomembno vlogo pri nadaljnjem načrtovanju in implementaciji rešitev.

3.1 Procesni model

Procesni model je formalni način predstavitve, kako nek poslovni svet deluje. Opisuje aktivnosti in njihove povezave med sabo. Lahko se predstavi z različnimi tehnikami, najpogostejše med njimi so petrijeve mreže, podatkovni diagrami tokov, BPMN, ... [6]. Dober procesni model ni lahko sestaviti. Ve-

lja pravilo, da je narediti dober model bolj umetnost kot znanost. Glavne lastnosti, ki jih mora vsebovati dober procesni model so naslednje:

- Slediti mora kaj se dejansko dogaja tekom procesa
- Prevzeti pogled nevtralnega zunanjega opazovalca, kateri gleda, kako je bil proces izveden in ugotovi izboljšave, ki so potrebne, da proces deluje bolj učinkovito
- Definira želeni proces in kako bi moral delovati
- Postavi pravila, smernice in obnašanje; če jim proces sledi bi morale zadostiti želenim rezultatom procesa.
- Poda razlago o racionalnosti procesa
- Razišče in ovrednoti več možnih smeri delovanja, ki temeljijo na racionalnih argumentih
- Definira točke za uporabo izvlečenih podatkov za analizo v poročilih[7]

Najpogostejše napake, ki se pojavljajo pri zasnovi procesnih modelov pa so:

- Model predstavlja idealizacijo realnosti. Načrtovalec procesa se osredotoči na želen oz. normalen način delovanja, torej model npr. pokrije samo 80% reprezentativnih primerov ostalih 20% pa zaradi netipičnega delovanja pozabi. Ponavadi pa ravno teh 20% ustvari večino problemov v sistemu. Razlogi za takšno poenostavljanje so v tem, da se zaradi nepopolnega razumevanja vseh primerov in pristranskosti, odvisni od vloge načrtovalca v organizaciji, naredi takšen model. Prav tako so ročno narejeni modeli ponavadi subjektivni in poenostavljeni zaradi lažjega razumevanja.
- Nezmožnost, da bi ustrezno zajeli človeško vedenje. Preprosti matematični modeli lahko zadostno opišejo delovanje ljudi za tekočim trakom, problem pa nastane, ko želimo opisati model z ljudmi, ki sodelujejo v večih procesih z različnimi prioritetami. Delavec, ki sodeluje

v večih procesih, mora razdeliti svoj čas med več teh procesov, zato je težko modelirati proces v izolaciji. Delavci pa prav tako ne delajo z enako hitrostjo. Modeli pa po navadi vzamejo fiksno razdelitev resursov in fiksno časovno okno za razpolago resursa.

- Model je v napačnem abstrakcijskem nivoju. Odvisno od vhodnih podatkov in vprašanj, ki jih želimo odgovoriti, mora biti izbran primeren abstrakcijski nivo, na primer model je preveč poenostavljen, da bi lahko odgovorili na relevantna vprašanja, ali pa je preveč podroben, da bi lahko razumeli dejansko vlogo in delovanje procesa.

To so samo nekateri od problemov, ki nastajajo pri zasnovi modela. Le izkušeni načrtovalci po navadi naredijo dober model, primeren za začetek implementacije ali ponovne implementacije. Nepravilni in slabi modeli lahko privedejo do nerazumevanja procesa in napačne implementacije. Ravno zato se pri procesnem rudarjenju zanašamo na dogodkovne podatke, se pravi podatke, ki že obstajajo o nekem procesu in na podlagi le-tega ustvarijo model. Vendar pa ne ustvarijo modela samo na eni ravni, lahko se pogleda model z večih različnih nivojev, na primer za 90% najbolj pogostejšimi primeri. Prav tako pa lahko razkrije, da ljudje v organizaciji ne delujejo kot stroji. Na eni strani lahko razkrije dosti neizkoriščenosti, na drugi strani pa razkrije fleksibilnost drugih delavcev pri različnem delu.[3]

3.2 Petrijeve mreže

Dober procesni model je težko narediti. Ogromno pomoči pa lahko pri tem pridobimo s procesnim rudarjenjem. Z algoritmi za iskanje procesov(npr. alfa algoritem, genetski algoritem, hevristično rudarjenje,...) lahko računalnik samodejno, glede na zapise v bazi generira procesni model. Obstaja več različnih možnosti prikaza procesnega modela. V programu Prom je najbolj razširjen prikaz s petrijevimi mrežami. Je pa predvsem odvisno od vrste uporabljenih algoritmov in želje predstavitve procesa. Ponavadi obstaja tudi

možnost samodejne pretvorbe v druge prikaze, kot so BPMN, UML diagrame, ...

3.2.1 Osnovni pojmi

Katerikoli dinamični sitem lahko opišemo z naslednjimi elementi:

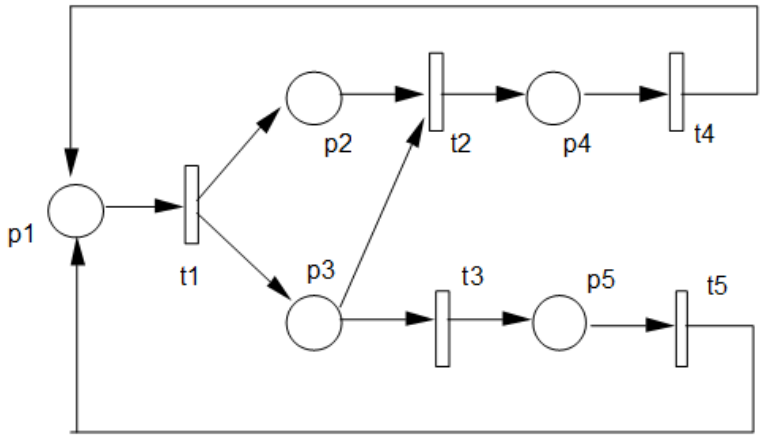
- Akcijami(T-transition)
- Pogoji(P-places)
- Enosmernimi povezavami med akcijami in pogoji(I,O-input, output)
- Žetoni

Grafično pa so ti elementi predstavljeni na naslednji način. Akcije predstavljamo s pravokotniki, pogoje s krogi, enosmerne povezave s puščico in žetone s piko.

Petrijeve mreže pa definiramo kot urejen četverček $C = (P, T, I, O)$, kjer sta $P = p_1, p_2, \ldots p_n, n \geq 0$ (končna množica mest) in $T = t_1, t_2, \ldots t_m, m \geq 0$ (končna množica prehodov), ter velja, da sta množici P in T tuji množici. I je vhodna funkcija, O pa je izhodna funkcija. Torej nam funkciji I in O definirata povezave med akcijami in pogoji.[26]

3.2.2 Petrijev graf

Petrijev graf je biparitetni graf, ki je sestavljen iz dveh tipov vozlišč. Za vozlišče vzamemo pogoj ali akcijo. Pogoje označimo s krogom, akcije pa s pravokotnikom ali črtico, vse skupaj pa zaključijo usmerjene povezave. Usmerjena povezava povezuje pogoj z akcijo ali akcijo s pogojem, ne more pa povezovati dveh enakih vozlišč npr. pogoj z pogojem ali akcijo z akcijo. Vsak pogoj pa lahko vsebuje žeton. Žeton označimo z črno piko v pogoju. Delu z žetoni v petrijevi mreži pravimo označevanje. Primer petrijevega grafa si lahko ogledamo na sliki 3.1 . [27]



Slika 3.1: Primer petrijeve mreže [27]

3.2.3 Označevanje in izvajanje v Petrijevi mreži

Označevanje v petrijevi mreži je dodeljevanje osnovnih postavk (žetonov) posameznim mestom v mreži. Število žetonov se lahko pri izvajanju petrijeve mreže spreminja. Označevanje o v petrijevi mreži je funkcija, ki mestom P priredi pozitivna cela števila N. Označevanje lahko predstavimo tudi z označitvenim vektorjem o. Označeno Petrijevo mrežo zapišemo kot $M = (P, T, I, O, o)$, pri čemer velja, da je $o = o_1, o_2, \ldots o_n, n = |P|$.

S tem, ko je petrijeva mreža označena, je možno tudi njeno izvajanje, v nasprotnem primeru pa to ni možno. Izvajanje mreže se izvede z vžigom izbranega prehoda t_i. Vžig se izvede tako, da se iz vhodnih mest odvzamejo ter na izhodna mesta dodajo žetoni. Vžig je možen le takrat, ko obstaja na vseh vhodnih mestih vsaj en žeton. Množico vseh možnih stanj petrijeve mreže pa imenujemo prostor stanj. Prav tako pa morajo veljati naslednja pravila. Vžig prehoda se izvede v idealnem času, čas vžiga prehoda je logičen in ne absolutni čas. V asinhronem primeru vžge prehod t_j takoj, ko je izbran, v sinhronem primeru vžge takrat, ko nastopi urin cikel. Če je izbranih več prehajanj istočasn,o se lahko izbere prehajanje glede na prioritetni sistem, naključno ali pa, če primer ni konflikten, lahko vžge več prehajanj hkrati. [8]

3.3 WorkFlow mreže

Posebna oblika petrijevih mrež je workflow mreža. Petrijevo mrežo lahko imenujemo tudi workflow mreža samo v primeru, da upošteva naslednja pravila:

- Obstaja samo en sam izvor procesa, označimo ga z i.
- Obstaja samo en sam ponor procesa, označimo ga z o.
- Vsako vozlišče je na poti iz i do o
- Ne obstaja reset povezava do ponora.

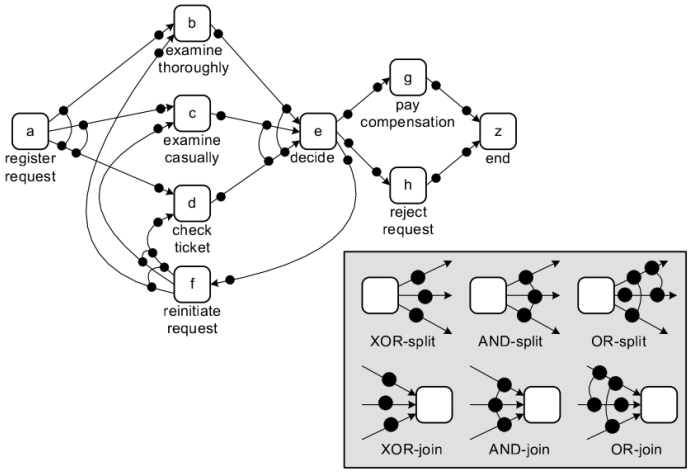
Ko sestavimo workflow mrežo, nas ponavadi najbolj zanima vprašanje "Ali je ta workflow mreža pravilno sestavljena?". Ker pa brez domene ne moremo odgovoriti na to vprašanje, lahko odgovorimo le na vprašanja kot so "Ali se mreža zaključi","Ali obstajajo kake mrtve zanke", in podobno. Iz tega potem izhaja pojem uglašenost WF mreže.[28]

Če vzamemo WF mrežo N z začetnim vozliščem i in izhodnim vozliščem o, potem je mreža N uglašena, če veljajo naslednji pogoji:

- Varnost: $(N, [i])$ je varna, ko pogoji ne morejo vsebovati več žetonov naenkrat.
- Pravilno končanje: za vsako označitev $M \in (N, [i]), o \in M$ velja $M = [o]$
- Možnost za dokončanje: za vsako označitev $M \in (N, [i]), [o] \in [N, M]$
- Odsotnost mrtvih zank. [3]

3.4 C-mreže

C-mreže definiramo kot graf, pri katerem vozlišča predstavljajo aktivnosti in povezave priložnostne odvisnosti. Vsaka aktivnost ima možna vhodna



Slika 3.2: Primer C-mreže [29]

in izhodna pravila. Če si kot primer ogledamo sliko 3.2, aktivnost a nima nobenega vhodnega pravila, saj je začetna aktivnost. Zato pa ima dve izhodni pravili, in sicer $\{b, d\}$ in $\{c, d\}$. To pomeni, da a sledi ali b in d oz. c in d. To povezavo označimo kot povezani piki v grafu. Za razliko od petri mrež pa pri c-mrežah nimamo pogojev.

C-mreža je definirana kot $C = (A, a_i, a_o, D, I, O)$ za katere velja:

- A je končni niz aktivnosti
- $a_i \in A$ je začetna aktivnost
- $a_o \in A$ je končna aktivnost
- $D \subseteq A \times A$ je relacija odvisnosti aktivnosti
- $I \in A$ definira vhodna pravila
- $O \in A$ definira izhodna pravila

tako da velja naslednje

- $D = \{(a_1, a_2) \in A \times A | a_1 \in \bigcup_{as \in I(a_2)}\}$

- $D = \{(a_1, a_2) \in A \times A | a_2 \in \bigcup_{as \in I(a_1)}\}$
- $\{a_i\} = \{a \in A | I(a) = \{\emptyset\}\}$
- $\{a_o\} = \{a \in A | O(a) = \{\emptyset\}\}$

Če želimo zapisati c-mrežo iz primera na sliki 3.2 dobimo naslednje vrednosti. $A = \{a, b, c, d, e, f, g, h, z\}$, kar predstavlja vse aktivnosti, $a = a_i$ je unikatna začetna aktivnost in $z = a_o$ unikatna končna aktivnost. Povezave predstavljajo odvisnostno relacijo med aktivnostmi in se jih označi kot $D = \{(a, b), (a, c), (a, d), (b, e), ..., (g, z), (h, z)\}$. Funkciji I in O pa določata pravila povezav. Za aktivnost a je $I(a) = \{\emptyset\}$, ter $O(a) = \{\{b, d\}, \{c, d\}\}$, potem $I(b) = \{\{a\}, \{f\}\}$ ter $O(b) = \{\{e\}, ...\}$, ter vse do $I(z) = \{\{g\}, \{h\}\}$, $O(z) = \{\emptyset\}$. [29]

3.4.1 Vzorčna matrika

Posebna predstavitev c-mrež je predstavitev z vzorčno matriko. Ta predstavitev nam pride zelo prav pri gradnji npr. genskega algoritma za iskanje procesov, saj predstavlja poenostavljeno predstavitev c-mrež. Kljub temu pa lahko brez problema pretvorimo matriko v c-mrežo ter nato, če želimo, še c-mrežo v petrijevo mrežo.

Vzorčna matrika predstavlja vzorčne odvisnosti med aktivnostmi. Če pogledamo na tabeli 3.1, vidimo da ni vzorčne odvisnosti med A in A saj je vrednost v matriki 0. Zato pa obstaja vzorčna odvisnost med A in B, to nam pokaže vrednost 1 v tabeli. Nadaljnji vnosi pa nam razkrijejo odvisnost med C in D za aktivnost A. Iz te matrike vidimo, da vsepovsod, kjer je vrednost ena, naredimo povezavo med aktivnostmi, saj nam to nadomesti funkcijo D v c-mrežah. Zadnji stolpec in prva vrstica pa nam povesta kakšna je logika med povezavami, se pravi za primer aktivnosti A je logika $B \vee C \vee D$ in glede na te vrednosti postavimo pike oz. loke v grafu c-mreže. To pa nam nadomesti funkciji I in O. Formalna definicija vzorčne matrike pa je naslednja:

	true	A	A	A	D	D	$E \wedge F$	$B \vee C \vee G$	
$\rightarrow$	A	B	C	D	E	F	G	H	$OUTPUT$
A	0	1	1	1	0	0	0	0	$B \vee C \vee D$
B	0	0	0	0	0	0	0	1	H
C	0	0	0	0	0	0	0	1	H
D	0	0	0	0	1	1	0	0	$E \wedge F$
E	0	0	0	0	0	0	1	0	G
F	0	0	0	0	0	0	1	0	G
G	0	0	0	0	0	0	0	1	H
H	0	0	0	0	0	0	0	0	true

Tabela 3.1: Primer vzorčne matrike [15]

Izrek 3.1 *Vzorčna matrika je sestavljena iz* $CM = (A, C, I, O)$*, kjer velja naslednje:*

- A *je končna množica aktivnosti*
- $C \subseteq A \times A$ *je vzorčna odvisnost relacij*
- $I \in A \rightarrow P(P(A))$ *je vhodna funkcija*
- $O \in A \rightarrow P(P(A))$ *je izhodna funkcija*

Za katero veljajo naslednja pravila:

- $C = \{(a_1, a_2) \in A \times A | a_1 \in \bigcup_{I(a_2)}\}$
- $C = \{(a_1, a_2) \in A \times A | a_2 \in \bigcup_{O(a_1)}\}$
- $\forall_{a \in A} \forall_{s,s' \in I(a)} s \cap s' \neq \emptyset \Rightarrow s = s'$
- $\forall_{a \in A} \forall_{s,s' \in O(a)} s \cap s' \neq \emptyset \Rightarrow s = s'$
- $C \cup \{(a_o, a_i) \in A \times A | a_o \overset{c}{\bullet} = \emptyset \wedge \overset{c}{\bullet} a_i = \emptyset$ *je močno povezan graf*

[15]

3.5 Ostali procesni modeli

Poleg omenjenih procesnih modelov obstaja tudi vrsta drugih procesnih modelov. Med zelo pogosto uporabljenimi so transition system, YAWL(Yet Another Workflow Language), BPMN(Business Process Modeling Notation), EPC(Event-Driven Process Chains),... Razlog zaradi katerega obstaja tako veliko število modelov, lahko označimo z besedo "predstavitvena pristranskost", to pomeni, da ima vsak model že v osnovi nekatere omejitve in pomanjkljivosti. Te pomanjkljivosti so lahko npr. nezmožnost predstavljanja sočasnosti(transition system, Markov model, flow chart), nezmožnost predstavljanja tihih akcij, nezmožnost predstavljanja podvojenih akcij(akcij z enakim imenom, zapisanih v dnevnikih dogodkov), nezmožnost prikaza ali razcepov, nezmožnost predstavitve hierarhije, ... Seveda pa obstajajo še veliko ostalih razlogov zakaj se uporablja tako veliko število modelov.[30]

Poglavje 4

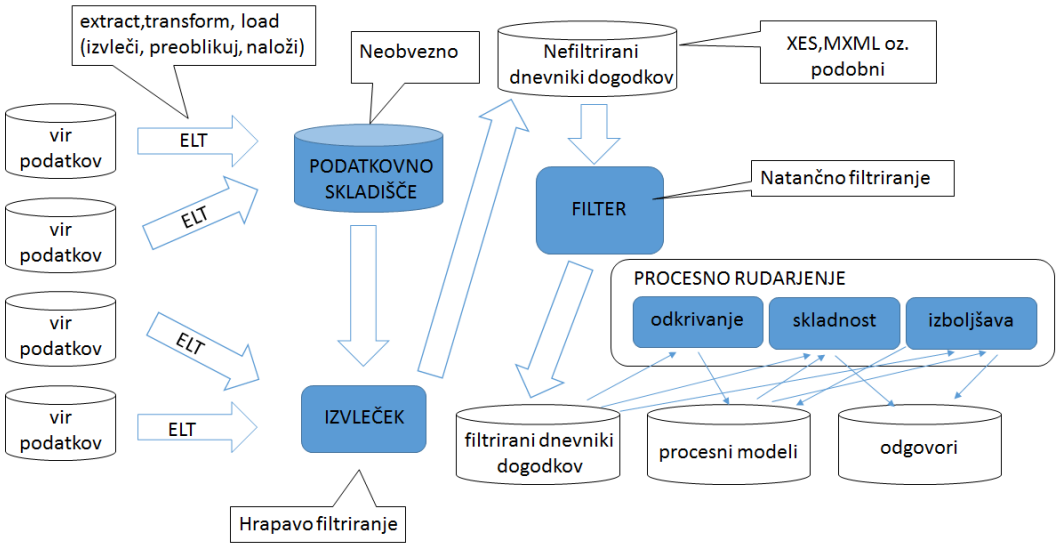
Dnevniki dogodkov

Procesno rudarjenje je nemogoče realizirati brez primernih dnevnikov dogodkov. Dnevniki dogodkov morajo zadostovati nekaterim kriterijem, ter slediti nekaterim pravilom. V realnosti je realizacija dnevnika dogodkov kar zapleten in težak postopek, kajti podatki so lahko razdrobljeni v različnih tabelah, ki se nahajajo v različnih podatkovnih bazah ter vsakemu podatku lahko manjka dobršen del informacije oziroma metapodatkov za primerno analizo.

4.1 Viri podatkov

Začetna pot zbiranja podatkov se večinoma začne na naslednji način. Najprej zberemo potrebne podatke, kajti podatki so po navadi zapisani v grobi obliki, torej niso primerni za analizo procesov z procesnim rudarjenjem. Viri podatkov so lahko od preprostih podatkovnih datotek, do excelovih preglednic, transakcijskih zapisov ter verjetno najpogostejših zapisov v podatkovni bazi. Problem pa ne nastane samo zaradi uporabe različnih virov podatkov. Podatki so lahko ne samo v različnih virih podatkov, ampak tudi v različnih tabelah, zato je potrebno precej napora, da izvlečemo relevantne podatke. Če vzamemo primer, je v celotni SAP implementaciji preko 10000 tabel.[3]

V splošnem se pa lotimo priprave podatkov za procesno rudarjenje na



Slika 4.1: Pregled postopka procesnega rudarjenja od podatkov do rezultatov [3]

naslednji način. Najprej moramo na virih podatkov izvesti ETL(extract, transform, load). Izraz prihaja iz poslovne inteligence in podatkovnega rudarjenja, pomeni pa, da moramo izvleči(exract) podatke iz zunanjih virov, jih preoblikovati(transform), da ustrezajo operacijskim potrebam ter nazadnje naložiti(load) v ciljni sistem. Ta je lahko podatkovno središče ali relacijska podatkovna baza. Cilj tega procesa je poenotenje podatkov na način, da jih lahko uporabimo za analize, poročila, predvidevanja, ... V naslednjem koraku je najbolj pomembno filtriranje in ocenjevanje podatkov. Pogosto ni problem sintetična pretvorba podatkov, ampak izbira primernih podatkov. Pomemben je odgovor na vprašanje: Katere tabele pretvoriti. V tem koraku se podatki shranijo v XES (eXtensible Event Steam) ali MXML (Mining eXtensible Markup Language) format.[9] Pomembno je, da pretvorimo samo tiste podatke, ki so relevantni pri naši analizi. Takšen dnevnik dogodkov je potem nefiltriran, kajti za nadaljnje delo, odvisno od tega katero vprašanje želimo natančno odgovoriti, izvedemo dokončno filtriranje. Nato nad tem dnevnikom dogodkov izvedemo odkrivanje, skladnost in izboljšave, ki so se-

stavni del procesnega rudarjenja. Za vse opisano pa moramo upoštevati še eno stvar, in sicer rezultati rudarjenja procesov sprožijo nova vprašanja in ta vprašanja lahko sprožijo potrebo po drugih virih podatkov, zato se ta proces pridobivanja podatkov iterativno ponavlja, dokler si ne odgovorimo na vsa zahtevana vprašanja.[3]

4.2 Dnevniki dogodkov

Dnevniki dogodkov so pomemben del pri sami analizi podatkov z rudarjenjem procesov. Če pogledamo sliko 4.1, se dnevnik dogodkov nahaja po izvlečku, oz. hrapavem filtriranju, torej je dnevnik dogodkov ena datoteka, zapisana v formatu XES ali MXML, ki mora upoštevati pravila. Obstajajo tri osnovna pravila, ki morajo veljati za dnevnike dogodkov. Ta pravila pa so naslednja:

1. ID primera(case ID) je identifikator procesa. Druga označba za ID primera je instanca procesa. Proces je sestavljen iz primerov. Pomemben je zaradi razlikovanja izvedbe istega procesa. Kaj natančno je primer ID-ja je odvisno od domene vsakega procesa(npr. v bolnišnici bi bil primer ID-ja pacient)

2. Aktivnost(Activity), vsak korak ali sprememba statusa v procesu mora biti poimenovana. Če je za vsako instanco procesa zapisano samo ena vrstica, potem podatki niso dovolj natančni. Podatki morajo biti na transakcijski ravni (moramo imeti dostop do zgodovine vsakega primera).

3. Časovni žig(timestamp), kajti rabimo vsaj en časovni žig, da razvrstimo podatke v pravilni vrstni red. Seveda jih pa rabimo tudi za določitev zakasnitev med aktivnostmi, ali pa ugotavljanje ozkih grl v samem sistemu. Zelo zaželeno je, da imamo zapisano kdaj se katera aktivnost začne in kdaj se tudi konča.[8]

Poleg teh pravil pa moramo za pravilno razumevanje dnevnikov dogodkov upoštevati še nekaj stvari. Sam proces je sestavljen iz primerov. Primer

ID primera	ID dogodka	Časovni žig	Aktivnost	Vir
1	1000	01.01.2013	(A) Order Goods	Peter
	1001	10.01.2013	(B) Receive Goods	Michael
	1002	13.01.2013	(C) Receive Invoice	Frank
	1003	20.01.2013	(D) Pay Invoice	Tanja
2	1004	02.01.2013	(A) Order Goods	Peter
	1005	03.02.2013	(B) Receive Goods	Michael
	1006	05.02.2013	(C) Receive Invoice	Frank
	1007	06.02.2013	(D) Pay Invoice	Tanja
3	1008	01.01.2013	(A) Order Goods	Louise
	1009	04.01.2013	(C) Receive Invoice	Frank
	1010	05.01.2013	(B) Receive Goods	Michael
	1011	10.01.2013	(D) Pay Invoice	Tanja
4	1016	15.01.2013	(A) Order Goods	Peter
	1017	20.01.2013	(C) Receive Invoice	Claire
	1018	25.01.2013	(D) Pay Invoice	Frank
5	1023	01.01.2013	(A) Order Goods	Michael
	1024	10.01.2013	(B) Receive Goods	Michael
	1025	13.01.2013	(C) Receive Invoice	Michael
	1026	20.01.2013	(D) Pay Invoice	Michael

Tabela 4.1: Primer dnevnika dogodkov [31]

je sestavljen iz dogodkov, in sicer na takšen način, da vsak dogodek pripada natanko enemu primeru. Dogodki v primeru so urejeni. Dogodki lahko imajo atribute. Primer najpogostejših atributov so aktivnost, čas, cena, resurs.

4.3 XES(eXtensible Event Stream)

Zelo pomemben del standardizacije pri procesnem rudarjenju je format, v katerem so zapisani podatki. Do sedaj je to vlogo opravljal MXML format, vendar pa so se mu začele kazati omejitve in pomanjkljivosti v tem, kakšne

podatke lahko in kakšne ne moremo shraniti. XES sloni na XML in je kratica za eXtensible Event Stream. Pri načrtovanju XES formata so bili najbolj pomembni naslednji cilji:

- Preprostost – ideja je, da predstavimo podatke na najbolj preprost možen način. XES naj bo preprost za ustvariti in členiti. Prav tako pa mora biti preprost za človeško branje.

- Fleksibilnost – XES mora biti sposoben zajeti dnevnike dogodkov iz katerega koli področja in ozadja, ne glede na izbrano domeno ali IT podporo opazujočega procesa, zato XES ni namenjen samo za uporabo v procesnem rudarjenju, temveč na splošno za beleženje dnevnikov dogodkov.

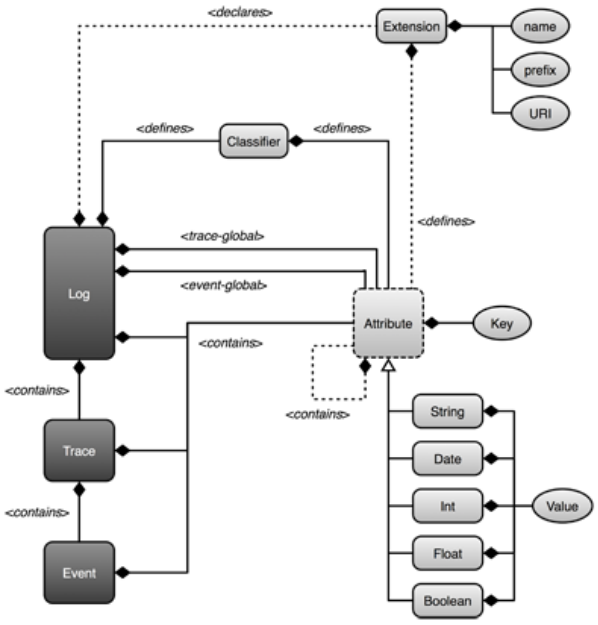
- Razširljivost – mora biti mogoče na preprost način omogočeno dodajanje standardu. Razširitve standarda morajo biti transparentne ter ohraniti kompatibilnost za nazaj, prav tako pa mora biti prilagodljiv za razširitve točno določenih domen ali posebnih zahtev.

- Izraznost – čeprav je cilj generalizacija formata, pa je trud v tej smeri, da dnevniki dogodkov izgubijo čim manj informacije kot je le mogoče. Zato morajo biti vsi informacijski elementi močno definirani, prav tako pa je zraven tudi privzeta metoda za vključitev človeško interpretiranega pomena podatkov.

Čeprav je XES dobil inspiracijo iz MXML se precej pogledih razlikuje od njega. Meta model za XES z UML razrednim diagramov vidimo na sliki 4.2.

XES ohranja privzeto strukturo dnevnika dogodka. Celotnemu procesu je enaka ena log datoteka, katera je sestavljena iz sledi(traces), se pravi posamezne instance izvedbe. Vsaka instanca tj. sled pa je sestavljena iz zaporedja dogodkov, prav tako pa lahko vsaka od teh treh konceptov vsebuje atribute, ki vsebujejo dejanske podatke o procesu.

Za razliko od MXML so sedaj atributi enakovredni, ne obstaja več posebnih polj, kot je na primer »originator«. Še bolj pomembno pa je, da so sedaj



Slika 4.2: Metamodel XES z UML diagramom [10]

atributi »strongly typed«. Atributi lahko vsebujejo string, date(timestamp), integer, floating-point ali boolean. Te spremembe močno povečajo izraznost formata, saj močno povečajo enostavnost shranjevanja meta podatkov in izvedbe procesa(kot je npr. cena aktivnosti).

Z ukinitvijo namenskih, posebnih polj za ime aktivnosti, se pojavi težava v tem, da ne vemo kaj natančno vsak atribut pomeni, zato pa se za ta namen uporabljajo razširitve(extensions). Razširitev definira število standardiziranih atributov za vsak nivo hierarhije, skupaj z njegovim tipom in posebnim ključem atributov.

Privzeto XES standard vsebuje številne standardne razširitve. Nekatere od teh so:

- Concept extension: Definira atribute za imena elementov(ime aktivnosti, id sledi,..)
- Time extension: Definira standardni atribut za opis datuma in časa, ko se je dogodek zgodil.

- Lifecycle extension: Definira življenjski model aktivnosti ter v katerem delu življenjskega cikla se ta nahaja(začetek, konec, nadaljevanje,. . .)
- Organizational extension: Definira standardne atribute za ime, vlogo in skupino resursa, ki je sprožila dogodek.

Če XES zapis uporablja te standardne razširitve, bodo njihovi atributi pravilno interpretirani s strani aplikacije, ki analizira te podatke. Če pa zapis uporablja svoje posebne domene, jih je prav tako možno uporabiti z definiranjem svojih posebnih razširitvenih domen. Prav tako pa je dovoljeno definirati svoje posebne atribute.

Popravek iz formata MXML je tudi možnost izbire ravni abstrakcije. Pri MXML si bil po navadi prisiljen dnevnik dogodkov razdeliti v več datotek. Pri XES pa je dodan koncept klasifikatorja dogodka. Klasifikator preprosto definira skupek atributov, ki dodajo identiteto dogodku. To pomeni, če imata dva dogodka enake vrednosti klasifikatorja za te atribute, se smatrata kot enakovredna. Zato sedaj, če imamo dnevnik dogodkov z več ravnmi abstrakcije, sedaj preprosto samo enkrat pretvorimo z vsemi pomembnimi informacijami v atributih dogodkov, ter dodamo klasifikator za vsak nivo abstrakcije.[11]

Poglavje 5

Algoritmi za odkrivanje procesov

Zelo pomemben del procesnega rudarjenja je odkrivanje in izgradnja procesnih modelov, se pravi, da glede na zapisane podatke v dnevniku dogodkov sestavi model procesa. V programu ProM se za predstavitev modelov najpogosteje uporabljajo petrijeve mreže oz. WF mreže, ki so izpeljanke petrijevih mrež. Za odkrivanje procesov obstaja veliko algoritmov. Najbolj osnoven algoritem je alfa algoritem, vendar se ga kot takega večinoma ne uporablja zaradi nekaj pomanjkljivosti, katere se delno ali v celoti rešijo z izpeljankami tega algoritma.

5.1 Alfa algoritem

Proces za odkrivanje oz. ponovno odkrivanje WF-mrež je razdeljen na tri dele. Predprocesiranje je prvi del algoritma. V tem delu sklepa razmerja med akcijami oz. aktivnostmi, ki se v WF mreži pretvorijo v akcije. Nato sledi procesiranje, v katerem se izvede dejanski alfa algoritem ter nazadnje še poprocesiranje.

5.1.1 Razmerja

Prvi del pri razumevanju algoritma je, da moramo postaviti razmerja med akcijami v delovnem toku. Ta razmerja bomo pozneje uporabili za iskanje pogojev in povezav med akcijami in pogoji.

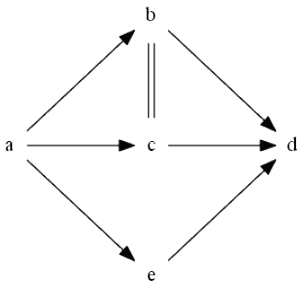
Definiramo naslednje relacije med akcijami v dnevniku dogodkov:

- Takojšen naslednik $x > y$, pomeni da aktivnosti x sledi y v dnevniku.
- Vzorčnost $x \rightarrow y$, pomeni x sledi y vendar pa y ne sledi x. Se pravi imamo zaporedje dveh dogodkov, vendar pa samo v eno smer.
- Vzporednost $x \parallel y$, pomeni $x > y$ in $y > x$. Tukaj pa velja da x sledi y in y sledi x aktivnosti.
- Nepovezanost $x \neq y$ pomeni x ne sledi y in y ne sledi x. Pomeni, da ne obstaja zapis v dnevniku, kjer bi aktivnosti x sledila aktivnost y in obratno.

Za vsak zapis v dnevniku dogodkov obstaja ena izmed opisanih relacij. Zapisu dnevnika dogodkov z opisom teh relacij pravimo odtis dnevnika. Vzemimo dnevnik dogodkov $L = [\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^2, \langle a, e, d\rangle]$, pripadajoč odtis dnevnika za ta primer pa dobimo v tabeli 5.1 .[12]

	a	**b**	**c**	**d**	**e**
a	#	$\rightarrow$	$\rightarrow$	#	$\rightarrow$
b	$\leftarrow$	#	$\parallel$	$\rightarrow$	#
c	$\leftarrow$	$\parallel$	#	$\rightarrow$	#
d	#	$\leftarrow$	$\leftarrow$	#	$\leftarrow$
e	$\leftarrow$	#	#	$\rightarrow$	#

Tabela 5.1: Primer odtisa dnevnika [32]



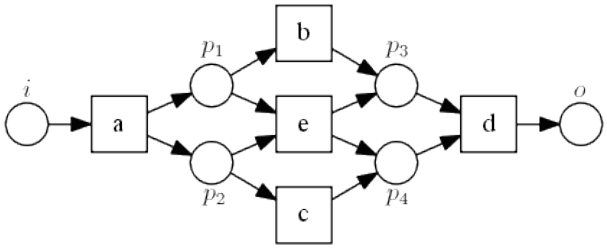
Slika 5.1: Primer vizualne predstavitve odtisa dnevnika. [12]

5.1.2 Delovanje alfa algoritma

Pri opisu algoritma privzamemo, da imamo dnevnik dogodkov L, T je seznam vseh akcij, T_1 je seznam vseh začetnih povezav, T_0 pa seznam vseh končnih povezav. Alfa algoritem deluje na naslednji način:

1. Iz dnevnika dogodka L izvleci vse povezave T
2. Določi T_1 in T_0, oz. seznam začetnih in končnih aktivnosti(akcij)
3. Najdi vse sete v paru (A, B) za katere velja
   (a) $t_a \in A$, ter je povezan preko prehoda p z $t_b \in B$
   (b) $\forall a \in A$ in $\forall b \in B$ za katerega velja $a \rightarrow b$
   (c) $\forall a_1, a_2 \in A$ velja $a_1 \# a_2$ in $\forall b_1, b_2 \in B$ velja $b_1 \# b_2$
4. Ko najdemo take sete parov, ohranimo samo maksimalne, se pravi tiste, s katerimi lahko povežemo maksimalno število elementov.
5. Povežemo vse elemente iz A z elementi iz B z enim pogojem $p(A, B)$,
6. ter na koncu še povežemo vse začetne povezave z začetkom in vse končne povezave z koncem.

Če vzamemo dnevnik dogodkov $L = [\langle a, b, c, d\rangle, \langle a, c, b, d\rangle, \langle a, e, d\rangle]$ lahko za lažjo predstavo, vizualno predstavimo odtis dnevnika, kot je tudi v tabeli 5.1



Slika 5.2: Primer dobljenega grafa z alfa algoritmom. [12]

na način, ki je na sliki 5.1. Nato poiščemo maksimalne sete, ki so prikazani v tabeli 5.2, ter na koncu še samo povežemo vse maksimalne sete med seboj in dobimo WF mrežo, ki je prikazana na sliki 5.2.[12]

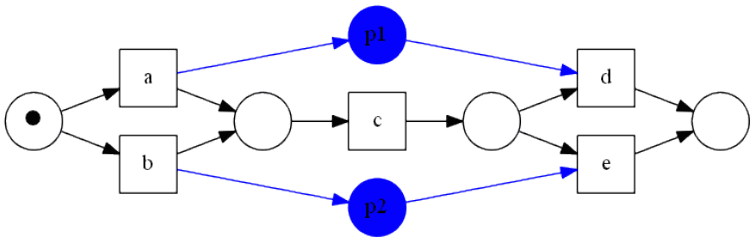
	A	B
(1)	$\{a\}$	$\{b, e\}$
(2)	$\{a\}$	$\{c, e\}$
(3)	$\{b, e\}$	$\{d\}$
(4)	$\{c, e\}$	$\{d\}$

Tabela 5.2: Primer maksimalnih setov. [12]

5.1.3 Omejitve alfa algoritma

Zaradi lastnosti iskanja povezav z alfa algoritmom, je ta deležen kar nekaj omejitev. Zaradi tega se v praksi ponavadi kot tak ne uporablja. Omejitve pa so naslednje:

- Zanka z dolžino 1. Če obstaja kratka zanka z dolžino 1, je alfa algoritem ne more odkriti. Razloge zakaj je ne, najdemo v definiciji, kajti mi iščemo sete v A in B, za katerega velja da mora biti b v množici A in b v množici B, hkrati pa mora veljati da sta $b\#b$, kar pa ni mogoče po definiciji iskanja z algoritmom alfa.



Slika 5.3: Primer ne-lokalne odvisnosti. [12]

- Zanka z dolžino 2. Prav tako ne more najti zank z dolžino 2. Razlog zakaj je ne najde, je v mišljenju algoritma, saj pri relaciji $b \parallel c$ misli, da sledi iz $b > c$ in $c > b$, vendar pa pri zankah dolžine 2 temu ni tako. Zanke z večjo dolžino od 2 pa lahko algoritem brez problema odkrije in ustvari model.

- Ne-lokalne odvisnosti, katere nastanejo zaradi omejitev v nekaterih procesih. Primer ne-lokalne odvisnosti lahko vidimo na sliki 5.3. To sta pogoja p_1 in p_2.[12]

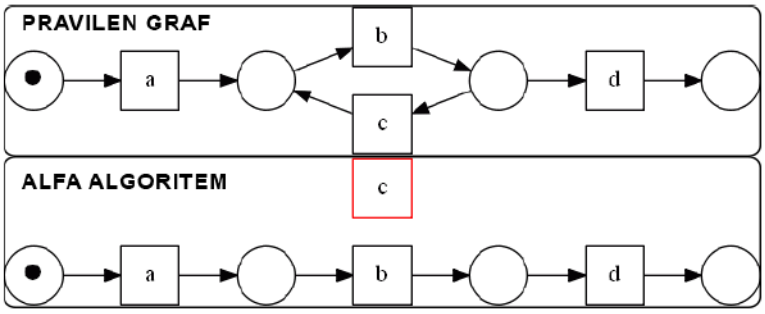
  - Te odvisnosti alfa algoritem ne zajame
  - Niso vidne v dnevnikih dogodkov
  - Te odvisnosti niso problem samo alfa algoritma, ampak velike večine algoritmov v procesnem rudarjenju

5.2 Alfa plus algoritem

Je izboljšava alfa algoritma in sicer v tem pogledu, da lahko odkrije kratke zanke dolžine 1 in 2. To pa naredi na naslednji opisan način.

5.2.1 Odkrivanje kratkih zank dolžine 2

Če si pogledamo primer na sliki 5.4, za dnevnik dogodkov $[\langle a, b, d\rangle, \langle a, b, c, b, d\rangle, \langle a, b, c, b, c, b, d\rangle$
vidimo da je razlog za neuspešno odkrivanje zank v nepravilni domnevi za



Slika 5.4: Primer iskanja zank dolžine 2 z alfa algoritmom. [12]

$b \parallel c$. Zaradi tega si moramo domisliti novo relacijo, da označimo to dogajanje. Najprej pa moramo definirati novo notacijo popolnosti za obvladovanje takšnih primerov. Za delovni tok N je dnevnik popoln, če je popoln in je dovolj informacij za detekcijo zank dolžine 2(moramo videti sekvence $..., a, b, a, ...a \neq b$, če obstajajo).

Kot omenjeno definiramo dve novi relaciji:

- $a \triangle b \iff$ velja za zapis $\ldots, a, b, a, \ldots$ v dnevniku dogodkov
- $a \Diamond b \iff$ obstaja sekvenca $\ldots, a, b, a, \ldots$ in $\ldots, b, a, b, \ldots$

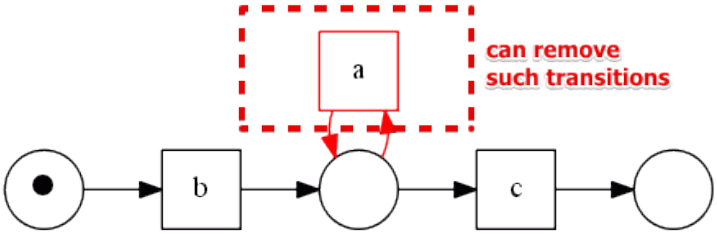
Popravimo pa tudi ostali relaciji na to, da izvzamemo zanke

- $a \rightarrow b \iff (a > b) \wedge (b \not> a \vee a \Diamond b)$
- $a \parallel b \iff (a > b) \wedge (b > a) \wedge (a \not\Diamond b)$

Pri tej definiciji pa moramo vseeno predvidevati, da zanke z dolžino 1 ni v dnevniku, kajti enak zapis lahko proizvede tudi zanka dolžine 1, zaradi tega pa problem zank dolžine ena odpravimo v predprocesiranju.[12]

5.2.2 Odkrivanje kratkih zank dolžine 1

Za odkrivanje zank dolžine 1 moramo najprej pogledati nekaj osnovnih lastnosti, ki veljajo.



Slika 5.5: Primer odstranitve akcije, ki je zanka dolžine 1 [12]

- Akcije, ki so zanke dolžine 1, ne morejo biti povezani na začetni ali končni pogoj.
- To akcijo lahko varno odstranimo iz grafa, saj je očitno, da ne bo vplivala na potek dogajanja in WF mreža bo ostala uglašena. Kot vidimo na sliki 5.5
- S takimi zankami se ukvarjamo v pred in po procesiranju.

Akcije, ki imajo zanke z dolžino 1 najdemo tako, da iščemo vzorec $\ldots, a, a, \ldots$ v dnevniku dogodkov.[12]

5.2.3 Potek in pomanjkljivosti

Alfa plus algoritem pa poteka tako, da najprej v predprocesiranju odstranimo vse akcije z zanko dolžine ena in dva ter izvedemo navaden alfa algoritem. V poprocesiranju pa dodamo omenjene akcije in na koncu dobimo pravilno WF mrežo.

Kljub tem dodatkom alfa plus algoritma, nam vseeno ostane še nekaj omejitev in pomanjkljivosti. Kot prva pomanjkljivost je ta, da privzamemo uporabo popolnih informacij. Se pravi, da predpostavljamo celovitost dnevnikov dogodkov. Naslednja pomanjkljivost je, da ne upoštevamo šuma. Šum pa se lahko pojavi kadarkoli in kjerkoli v zapisanih podatkih. To pa izhaja iz tega, ker alfa algoritem ne upošteva število pojavitev določenega zaporedja dogodkov. [12]

5.3 Hevristično rudarjenje

Naslednji algoritem, ki ga lahko uporabimo za procesno rudarjenje, je hevristični algoritem. Prednost pred alfa in alfa plus algoritmoma je v tem, da ta upošteva število pojavitev določenih aktivnostih, s tem pa tudi poskrbi za možen šum v podatkih, kajti sama ideja je v tem, da tistih poti, ki se zelo redko pojavijo ne vključimo v sam model.

5.3.1 Opis in prednosti hevrističnega rudarjenja

Rezultat hevrističnega rudarjenja se po navadi ponazori z C-mrežami, ki sem jih razložil že v prejšnjem poglavju. Prednost hevrističnega rudarjenja pred alfa algoritmom je v tem, da hevristično rudarjenje upošteva pogostost pojavljanja zaporedja nekih aktivnostih, prav tako ima alfa algoritem tudi pogoj, da ne dovoli preskakovanja aktivnosti. Hevristično rudarjenje pa s tem nima problemov. Prav tako pa lahko določimo, do kakšnega nivoja naj zanemari šum v dnevniku dogodkov.[13][3]

5.3.2 Potek hevrističnega rudarjenja

C-mreže se predstavi z $C = (A, a_i, a_o, D, I, O)$, pri čemer je A končna množica aktivnosti, a_i je začetna aktivnost, a_o je končna aktivnost, D je odvisnostna relacija, I je množica vhodnih aktivnosti, O je množica izhodnih aktivnosti. Zato najprej poiščemo vse aktivnosti v dnevniku dogodkov. S tem smo ugotovili A, a_i in a_o lahko prav tako poiščemo brez večjih problemov. Naslednji korak, ki ga moramo narediti pa je nastaviti odvisnostne relacije med vsemi pari aktivnosti. Odvisnostne relacije naredimo tako, da naredimo tabelo, v katero vpišemo kolikokrat je $x >_L y$. Se pravi preštejemo kolikokrat x sledi y. Za primer vzemimo:

$$L = [\langle a, e\rangle^5, \langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^{10}, \langle a, b, e\rangle, \langle a, c, e\rangle, \langle a, d, e\rangle^{10}, \langle a, d, d, e\rangle^2, \langle a, d, d, d, e\rangle]$$

Ko opravi prvi korak na našem primeru dobimo odvisnostne relacije, ki si jih lahko pogledamo na tabeli 5.3.

$>_L$	a	b	c	d	e
a	0	11	11	13	5
b	0	0	10	0	11
c	0	10	0	0	11
d	0	0	0	4	13
e	0	0	0	0	0

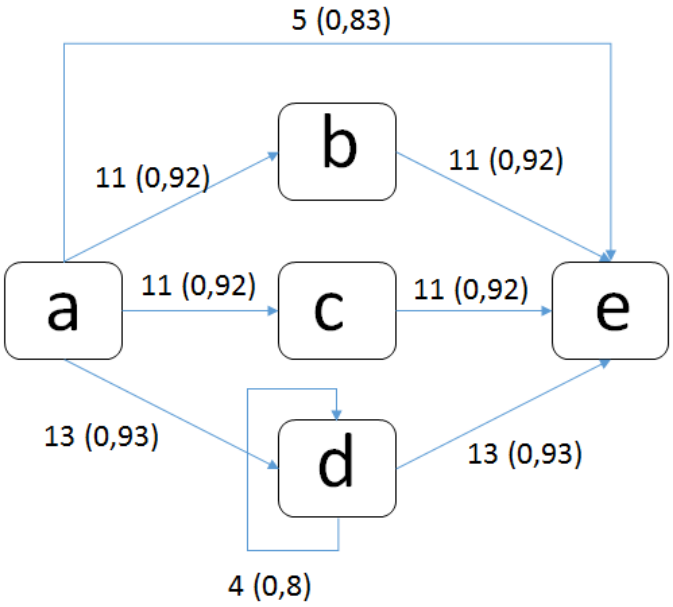
Tabela 5.3: Primer odvisnostne relacije [3]

Ko izračunamo odvisnostne relacije, sledijo izračuni vrednosti odvisnosti med akcijami. Za to pa uporabimo enačbo 5.1 . Po uporabi te enačbe dobimo nove vrednosti za vsako vrstico tabele 5.3. Na primer za aktivnost a dobimo vrednosti v relaciji z a dobimo $\frac{0}{0+1} = 0$ v relaciji z b dobimo $\frac{11-0}{11+0+1} = 0,92$ potem z c dobimo $\frac{11-0}{11+0+1} = 0,92$ z d je $\frac{13-0}{13+0+1} = 0,93$ in tako dalje dokler ne dobimo vrednosti za vse v tabeli.

Izrek 5.1

$$|a \Rightarrow b| = \begin{cases} \frac{|a>_L b|-|b>_L a|}{|a>_L b|+|b>_L a|+1}, & a \neq b \\ \frac{|a>_L a|}{|a>_L a|+1}, & a = b \end{cases} \tag{5.1}$$

Za obe tabeli potem določimo prag, katerega moramo upoštevati, da se med dvema aktivnostima ustvari povezava. Na našem primeru, če vzamemo prag 5 za $|>_L|$ in 0,7 za $|\Rightarrow_L|$ dobimo povezave na grafu, kot prikazuje slika 5.6 . Ko smo končali te postopke smo s tem dobili (A, a_i, a_o, D) vrednosti C-mreže. Manjkata nam še I in O. To pa dobimo tako, da preštejemo vse pojavitve povezav med njimi. Če si pogledamo na primeru na sliki 5.7 vidimo, da se akcija a pojavi 40-krat, akcija b 21 krat, akcija c tudi 21 krat, akcija d 17 krat in akcija e 40-krat. Potem pogledamo, koliko krat sledijo zaporedoma aktivnosti, ki imajo med seboj povezavo. Se pravi preverjamo povezave, ki se vzporedno izvedejo za naš primer. To pomeni, da preverjamo kombinacije med akcijami (a, e, b, c, d). V našem primeru ugotovimo, da se zaporedno izvajata v 20 primerih (a, b, c) medtem ko pa v 2 primerih temu ni tako. Ker pokrivamo večino med b in c naredimo lok, da se izvajata sočasno.



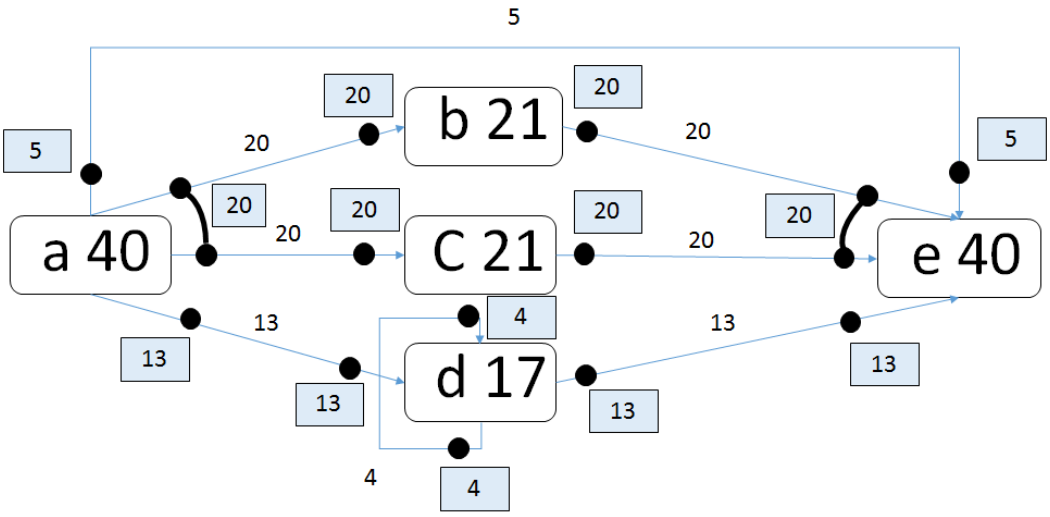
Slika 5.6: Primer praga z vrednostmi 5 za $| >_L |$ in 0,7 za $| \Rightarrow_L |$ [3]

Na ta način potem naredimo za celotni graf, dopolnimo I in O ter zaključimo našo C-mrežo.

Hevristično rudarjenje uporabimo takrat, ko imamo dejanske podatke z majhnim številom dogodkov ali pa, ko rabimo petrijevo mrežo za nadaljnje raziskave.[13][3]

5.4 Gensko procesno rudarjenje

Genski algoritmi so prirejeni načini iskalnih metod, ki poizkušajo posnemati delovanje procesa evolucije. Taki algoritmi se začnejo s prvotno(initial) populacijo posameznikov(v našem primeru s procesnim modelom). Nato se populacija razvija z izbiro najboljših(fittest) posameznikov. Te najboljše posameznike nato križamo(crossover) med seboj, da dobimo nove posameznike. Na koncu pa te posameznike še mutiramo(dodamo naključne spremembe). Ta proces ponavljamo, dokler ne zadostimo našim zahtevam.[14]



Slika 5.7: Grafična predstavitev izračuna I in O v C-mreži[3]

Če želimo uporabiti gensko rudarjenje, moramo biti zmožni predstaviti posameznike. Vsak posameznik mora biti del možnega procesnega modela in obenem enostaven za obdelavo. Začetno je bil namen predstavitve genskega rudarjenja s Petrijevimi mrežami, vendar pa se je izkazalo, da niso najboljše za tak način dela. Glavni razlog za to je, da ne moremo dobiti pogojev iz dnevnikov dogodkov. Dogodki v dnevniku dogodkov se nanašajo zgolj na akcije, iz tega pa sledi, da je dosti težje generirati začetno populacijo ter definirati križanje in mutacijo. Prav tako pa so problemi z in/ali povezavami. Zaradi tega je bila pri gradnji tega algoritma uporabljena vzorčna matrika. [15]

5.4.1 Delovanje algoritma

Cilj prvega dela algoritma je narediti začetne populacije. To se naredi tako, da se naredijo naključne vzorčne matrike. Vzorčna matrika je sestavljena iz A – seznam aktivnosti, C - vzorčne relacije, I – vhodna funkcija, O – izhodna funkcija. Pri kreiranju populacije pa velja, da imajo vse populacije enake aktivnosti. S tem smo že definirali množico A, saj je za vso populacijo enaka.

I in O množica sta naključno sestavljeni za vsak primer v populaciji. Za izračun C oz. vzorčne relacije pa se uporablja hevristika za mero odvisnosti. Motivacija za tem je preprosto v tem, če se pojavi v dnevniku dogodkov zapis t_1, t_2, je zapis t_2, t_1 pogosto le kot izjema. Obstaja pa velika verjetnost, da je med t_1 in t_2 vzorčna odvisnost. Rezultat vsega tega je v tem, da ima lahko začetna populacija katerega koli posameznika v iskalnem prostoru definiranega z začetnimi aktivnostmi.

Naslednji del, ki se ga moramo lotiti je natančnost izračuna. Če posameznik v tej populaciji pravilno opiše vedenje v dnevniku dogodkov, potem je natančnost tega posameznika visoka. V našem primeru je natančnost močno povezana s številom pravilno razčlenjenih sledi v dnevniku dogodkov. Moramo pa upoštevati, da ne moremo pričakovati popolno prilagajanje dnevnika dogodkov procesnemu modelu, to pa je zaradi šuma v sledi, katerega ne moremo upoštevati v želenem modelu. Natančnost, ki jo želimo doseči z genskim procesnim rudarjenjem izračunamo po formuli 5.2.

Izrek 5.2 *Formula za izračun natančnosti genskega procesnega rudarjenja*

$$fitness = 0,4 \times \frac{allParsedActivities}{numberOfActivitiesAtLog} + 0,6 \times \frac{allProperlyCompletedLogTraces}{numberOfTracesAtLog} \tag{5.2}$$

Kjer $numberOfActivitiesAtLog$ pomeni število vseh aktivnosti, $numberOfTracesAtLog$ število vseh zapisov v dnevniku, $allParsedActivities$ so vse aktivnosti, ki se lahko sprožijo brez umetnega dodajanja žetonov, $allProperlyCompletedLogTraces$ je pa število sledi, ki so bile pravilno razvrščene.

Nato sledijo genske operacije kot so elitizem, križanje in mutacija. Elitizem pomeni, da je samo odstotek najbolj natančnih oz. najmočnejših posameznikov uporabljen za naslednjo generacijo. Križanje ustvari nove posameznike s tem da križa najboljše posameznike v nove potomce. Se pravi križanje vzame najboljše dele najboljših posameznikov in jih med sabo križa z željo, da ustvari še boljše potomce. Algoritem se ustavi v naslednjih primerih:

- Najde posameznika, katerega natančnost je 1.

- Izračuna n generacij, za katere velja da je n največje možno število generacij.

- Najboljši posameznik se ni spremenil že $\frac{n}{2}$ generacij zaporedoma.

Če nobeden od teh pogojev ne velja, se generacije ustvarjajo po naslednjem pravilu:

VHOD: trenutna populacija, stopnja elitizma, stopnja križanja in stopnja mutacije

IZHOD: nova populacija

1. Skopiraj najboljše posameznike iz prejšnje populacije in jo dodaj novi populaciji

2. Dokler lahko še ustvarjamo posameznike:

   (a) Z tournament selection izberi $parent_1$

   (b) Z tournament selection izberi $parent_2$

   (c) Izberi naključno število r med 0(vključujoč) in 1(izključujoč)

   (d) Če je r manj kot stopnja križanja, potem $parent_1$ in $parent_2$ ter s tem dobiš dva potomca, v nasprotnem primeru pa $offspring_1$ je enak $parent_1$ in $offspring_2$ enak $parent_2$

   (e) Mutiraj $parent_1$ in $parent_2$

   (f) Dodaj $offspring_1$ in $offspring_2$ v novo populacijo

3. Vrni novo populacijo

Tournament selection: je način izbire starša, pri katerem algoritem naključno izbere pet posameznikov in med njimi izbere najboljšega(najbolj natančnega glede na formulo 5.2).[15]

5.5 Inductive miner

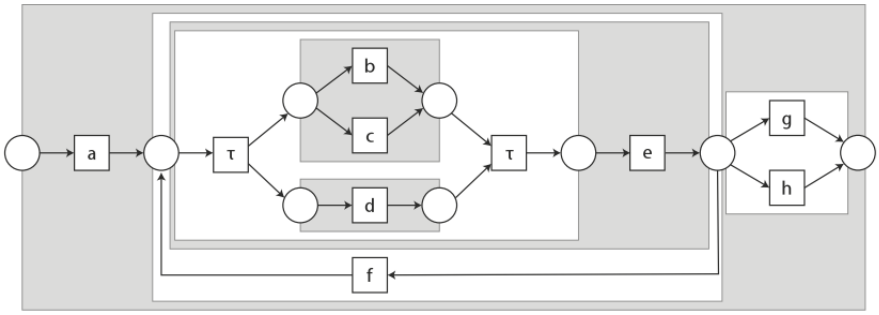
Kateri procesni model je najboljši, se tipično pogleda z različnimi preverjanji kakovosti. Predvsem je odvisno, kakšno kakovost želimo zagotoviti modelu. Ena izmed pomembnih kakovosti je soundness modela. Model šteje kot soundness takrat, ko je možno izvesti vse stopnje v procesu ter vedno dosežemo neko končno stanje. Naslednje možno preverjanje kakovosti je natančnost. Popolnoma natančen model lahko izvede vse zapise v dnevniku dogodkov. Obstaja pa še vrsta drugih kriterijev. Posebnost inductive minerja pred drugimi algoritmi je, da lahko zagotovi soundness ter obenem doseže zahtevano stopnjo natančnosti. Inductive miner uporablja algoritem deli in vladaj.[36][37]

5.5.1 Algoritem

Predpostavljamo, da imamo podano skupek aktivnosti A, ter dnevnik dogodkov L. Velja da je dogodek, ena aktivnost v dnevniku dogodkov $e \in A$. Sled t pa je možno prazno zaporedje dogodkov. Velja da je velikost dnevnika enaka $||L|| = \sum_{t \in L} |t|$.

Osnova za delovanje algoritma je procesno drevo. Procesno drevo je kompaktna abstraktna predstavitev bločne strukture workflow mreže. Je korensko drevo, v katerem so kot listi predstavljene aktivnosti, vsa ostala vozlišča pa so predstavljena kot operatorji. Drevo formalno predstavimo rekurzivno. Skupek aktivnosti predstavimo kot A in skupek operatorjev kot $\oplus$. Simbol τ pa predstavlja skrite aktivnosti in velja $\tau \notin A$. Prav tako bomo pa uporabili naslednje operatorje.

- operator $\times$ pomeni ekskluzivno izbiro med dvema poddrevesoma
- operator $\rightarrow$ pomeni zaporedno izvršitev vseh poddreves
- operator $\vee$ pomeni vzporedno izvrševanje poddreves
- operator $\hookrightarrow$ pa pomeni zanko na poddrevo



Slika 5.8: Primer razdelitve procesa na procesno drevesno [36]

Če si pogledamo na primeru iz slike 5.8 dobimo naslednji zapis procesnega drevesa: →(a,↪(→ (∨(×(b, c), d), e), f), ×(g, h)). Samo delovanje algoritma pa poteka tako, da imamo podan skupek pravil $\oplus$ ter definiramo ogrodje B za odrivanje procesnega modela, z uporabo deli in vladaj algoritma. Če imamo podan dnevnik dogodkov L, potem B išče možna razbitja L v manjše $L_1, ..., L_n$, ki skupaj z operatorji, ki so v $\oplus$, lahko ponovno zgradijo L. Potem se rekurzivno izvaja algoritem na najdenih razdelitvah ter vrne kartezični produkt najdenih modelov. Rekurzija se zaustavi, ko se L ne more bolj razdeliti. Manjši popravek moramo pri tej ideji narediti edino pri delitvi L, saj pri strogi delitvi nekateri modeli ostanejo neodkriti, na primer za neopazovane aktivnosti. Tako L razdelimo v podbloke, ki imajo enako velikost kot L, vendar pa se tudi to lahko zgodi samo tolikokrat. Zato predstavimo števec ϕ, ki se mora zmanjšati, ko je nemogoče enakomerno zmanjšati dnevnik L. Z drugimi besedami je to število nevidnih aktivnosti τ, sam algoritem pa je na sliki 5.9.[36]

5.6 Fuzzy miner

Je eden izmed najmlajših algoritmov za izgradnjo procesnih modelov iz dnevnikov dogodkov. Je pa prvi algoritem, katerega namen je analiza procesov z velikim številom aktivnostmi in zelo ne-strukturiranim delovanjem procesa.

function $B_{select}(L, \phi)$
  if $L = \{\epsilon\}$ **then**
    $base \leftarrow \{\tau\}$
  else if $\exists a \in \Sigma : L = \{\langle a \rangle\}$ **then**
    $base \leftarrow \{a\}$
  else
    $base \leftarrow \emptyset$
  end if
  $P \leftarrow select(L)$
  if $|P| = 0$ **then**
    if $base = \emptyset$ **then**
      return $\{\circlearrowleft(\tau, a_1, \ldots, a_m)$ where $\{a_1 \cdots a_m\} = \Sigma(L)\}$
    else
      return $base$
    end if
  end if
  return $\{\oplus(M_1, \ldots, M_n) | (\oplus, ((L_1, \phi_1), \ldots, (L_n, \phi_n))) \in P \wedge \forall i : M_i \in B(L_i, \phi_i)\} \cup$ $base$
end function

Slika 5.9: Inductive miner algoritem [36]

Kot izhod nam ta algoritem poda Fuzzy model, katerega pa ni možno pretvoriti v druge procesne modele. Za razliko od hevrističnega rudarjenja pa tukaj lahko skrijemo manj pomembne aktivnosti ali pa jih združimo v skupine, katere potem v modelu ne vidimo. [13]

5.7 Regijsko-temeljen algoritem

Za to metodo iskanja procesnega modela je značilno da dobimo petrijevo mrežo, katera pretirano približuje obnašanje zapisov v dnevniku dogodkov. Najpomembnejša lastnost te metode je, da vrne mrežo z najmanjšim obsegom, kateri še vedno pokrije obseg dnevnika dogodkov. Algoritem je sestavljen iz dveh delov. Prvi del je dobiti transition sistem, drugi del pa je ta sistem pretvoriti v petrijevo mrežo. V prvem delu določimo želeno abstrakcijo v dnevniku dogodkov. V drugem delu pa to abstrakcijo predstavimo na želen sistem, s sintezo.[16]

Poglavje 6

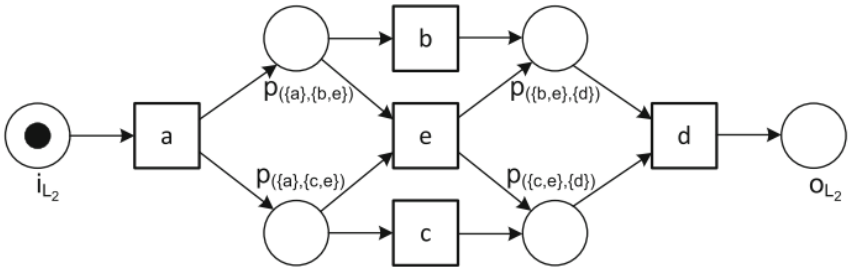
Preverjanje skladnosti

Naslednji možen način uporabe procesnega rudarjenja je preverjanje skladnosti. Pri tem načinu imamo kot vhod podan procesni model in dnevnik dogodkov. Nato pa preverjamo skladnost modela s podatki, kot rezultat pa dobimo odstopanja podatkov od modela. Pridobljene ugotovitve pa lahko uporabimo za poslovno poravnavo(analizo procesne uspešnosti in izboljšanja), revizijo(najdbo goljufij in nepravilnosti) ali pa za analizo procesnih iskalnih algoritmov. Obstaja veliko različnih načinov in metrik za testiranje skladnosti. K vsemu temu pa lahko skladnost merimo na različnih nivojih. Možni nivoji so meritev na ravni primera, dogodka, sledi in omejitev. Skladnost se pa lahko testira online(med izvajanjem procesov) ali offline(po končanju procesov).

Obstajata dva možna načina za testiranje skladnosti, in sicer Data Analysis in Conformance testing. Data analysis se osredotoča na primerjanje modela z modelom, medtem ko Conformace Testing neposredno primerja model s podatki v dnevniku dogodkov. [17]

6.1 Token Replay

Najenostavnejša metrika za izmero natančnosti je izračun popolnoma prilegajočih sledi.



Slika 6.1: Primer grafa [18]

Če si pogledamo na primeru iz slike 6.1 ter ga primerjamo z dnevnikom dogodkov $L = [\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^3, \langle a, e, d\rangle^2, \langle a, d\rangle, \langle a, e, e, d\rangle]$ ima po prilegajočih sledeh natančnost 0,8. To pa zaradi tega, ker je 8 od 10 sledi možno rekonstruirati s pripadajočim modelom. Vendar pa tega naivnega pristopa ne moremo uporabiti pri bolj kompleksnejših in realnih primerih, saj ima eno veliko pomanjkljivost, ne more razločiti med »skoraj se ujema« s tistimi, ki se popolnoma ne prilegajo. Zato rabimo neko mero, s katero bomo merili na ravni dogodkov ne pa na ravni sledi. Se pravi rabimo nekaj, s čimer bomo kljub napaki nadaljevali in na koncu zabeležili število manjkajočih žetonov in koliko žetonov bo ostalo v sistemu. Tukaj pa uporabimo mero token replay. Za predstavitev ideje je najlažje, če pogledamo na primeru. Želimo na primer izvesti naslednjo zaporedje dogodkov: (a, b, c, d) na grafu iz slike 6.1. Uporabimo štiri mere:

- p(ustvarjene žetone)
- c(porabljene žetone)
- m(manjkajoči žetoni)
- r(preostali žetoni)

Na začetku je $p = c = 0$ in vsi pogoji so prazni. Potem okolje ustvari en žeton na začetku sistema. Zato je $p = 1$. Nato se sproži akcija a. To je

mogoče, saj se porabi en žeton in se ustvarita dva na izhodu a, kot poteka proženje pri Petrijevi mreži. Zato se p poveča za 2 in je sedaj $p = 3$, $c = 1$ nato sledi sprožitev akcije b, katera poda $p = 4, c = 2$. Po sprožitvi c pa je $p = 5, c = 3$. Nato pa še sledi d, $p = 6, c = 5$. Na koncu pa še okolje požre en žeton in je rezultat $p = 6, c = 6$. Očitno je da okolje porabi toliko žetonov kot jih proizvede. Se pravi da se ta sled popolnoma izvede($m = r = 0$). Natančnost sledi pa je definirano po formuli 6.1.

Izrek 6.1 *Formula za izračun natančnost sledi pri token replay*

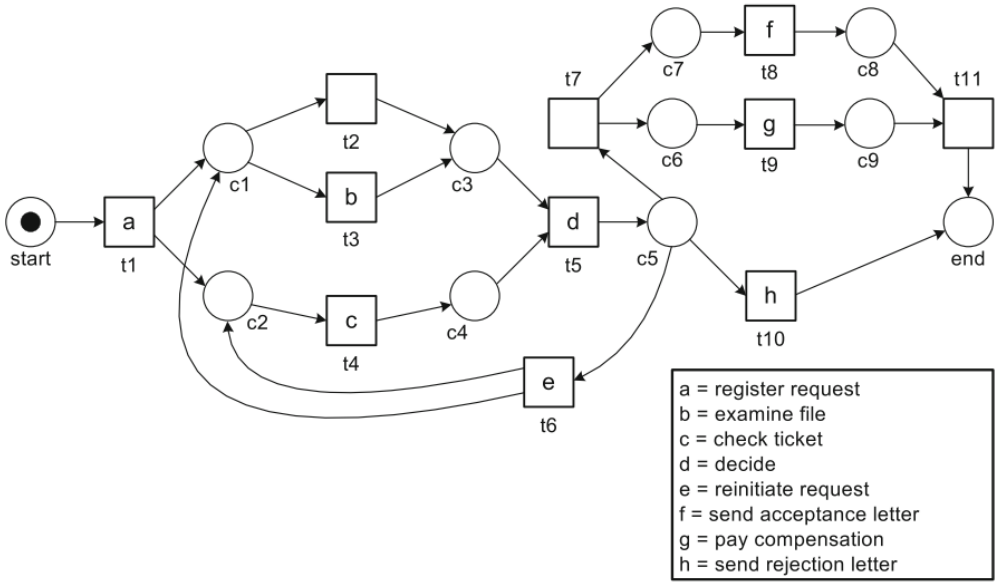
$$fitness(\sigma) = \frac{1}{2} \times \left(1 - \frac{m}{c}\right) + \frac{1}{2} \times \left(1 - \frac{r}{p}\right) \tag{6.1}$$

Ko vstavimo vse parametre prejšnjega primera v formulo 6.1 dobimo rezultat 1, kajti noben žeton ne manjka ali bi ostal v sistemu. Sedaj pa poglejmo primer, kjer pa se vse ne izide. Izračunajmo natančnost prileganja sledi (a, d). Na začetku je $p = c = 0$, spet ustvarimo žeton na začetku sistema $p = 1$. Prva akcija a se lahko sproži in tako nastane $p = 3,\ c = 1,\ m = 0\ in\ r = 0$. Sedaj poizkušamo izvesti drugo akcijo d. To ni mogoče, ker akcije d ne moremo aktivirati. Če želimo aktivirati d moramo vstaviti dva manjkajoča žetona na vsak vhod $d - ja$. Tako dobimo naslednje številke $p = 4,\ c = 3,\ m = 2\ in\ r = 0$. Na koncu okolje porabi en žeton, vendar pa nam ostaneta 2 v sistemu. Tako je končni rezultat $p = c = 4$ in $m = r = 2$. Če vstavimo številke v formulo 6.1 dobimo rezultat, da je natančnost te sledi 0,5.

S tem smo izračunali natančnost za en primer, podobno izračunamo natančnost za celotno skupino primerov. Preprosto vzamemo povprečje vseh oz. uporabimo formulo 6.2.[18]

Izrek 6.2 *Formula za izračun natančnost vseh sledi pri token replay [18]*

$$fitness(L) = \frac{1}{2} \times \left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times m_\sigma}{\sum_{\sigma \in L} L(\sigma) \times c_\sigma}\right) + \frac{1}{2} \times \left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times r_\sigma}{\sum_{\sigma \in L} L(\sigma) \times p_\sigma}\right) \tag{6.2}$$



Slika 6.2: Primer petrijeve mreže [18]

6.2 Cost-based alignments

Obstaja veliko načinov za izračun in meritev natančnosti. Token replay, s katerim štejemo manjkajoče, obstoječe, narejene in porabljene žetone, ima nekaj omejitev. Na primer za akcije, ki imajo enako poimenovane več različnih aktivnostih ali pa za skrite aktivnosti nastanejo vse mogoče komplikacije. Kot je na primer katero pot vzeti, če je omogočenih več akcij z enako oznako. Ali pa če ima sistem zelo slabo natančnost, je ta poplavljen z žetoni in je ocena natančnosti zelo optimistična. Cost-based alignments prinese bolj robusten in fleksibilen pristop k preverjanju skladnosti.

Za izračun natančnosti, moramo najprej »poravnati« sledi v dnevniku dogodkov s sledmi procesnega modela. Najlažje pogledamo ta postopek na primeru. Imamo podan graf na sliki 6.2. Poglejmo si primer poravnave sledi:

$L = [\langle a, c, d, f\rangle^{10}, \langle a, c, d, h\rangle^{5}, \langle a, b, c, d, e, c, f, g, f\rangle^{5}]$, na sliki 6.2.

Prva vrstica vsake poravnave pripada »premikom v dnevniku«, naslednji

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & c & \gg & d & \gg & f & \gg & \gg \\ \hline a & c & \tau & d & \tau & f & g & \tau \\ t1 & t4 & t2 & t5 & t7 & t8 & t9 & t11 \\ \hline \end{array} \qquad \gamma_5 = \begin{array}{|c|c|c|c|c|c|} \hline a & c & \gg & d & c & h \\ \hline a & c & \tau & d & \gg & h \\ t1 & t4 & t2 & t5 & & t10 \\ \hline \end{array}$$

$$\gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & \gg & d & e & c & \gg & d & \gg & g & f & \gg & h \\ \hline a & b & c & d & e & c & \tau & d & \tau & g & f & \tau & \gg \\ t1 & t3 & t4 & t5 & t6 & t4 & t2 & t5 & t7 & t9 & t8 & t11 & \\ \hline \end{array}$$

Slika 6.3: Primer poravnave sledi pri Cost-based alignments [18]

dve vrstici pa »premikom v modelu«. Premiki v modelu so predstavljeni z akcijami in njenimi oznakami, saj lahko obstaja več akcij z enako oznako. Pri poravnavi pa najprej izvedemo prvi stolpec s prvo akcijo tako v premiku modela kot v premiku dnevnika. V primeru, da ne moremo rekonstruirati sledi vstavimo znak »>><<, če pa obstaja skrita akcija, potem pa vstavimo znak »τ«. Na ta način sledimo zapisu v dnevniku in preverjamo, kje obstaja zapis v modelu in kje obstaja skrivna akcija, ter tako dobimo poravnane modela.

En tak premik lahko zapišemo s parom $(x, (y, t))$, kjer se prvi element nanaša na zapis v dnevniku drugi pa v modelu. Na primer za prvi primer zapisa v dnevniku je par $(a, (a, t_1))$, to pomeni, da oba dnevnik in model naredita »a« premik in premik v modelu je povzročen s pojavitvijo akcije t_1. Za izračun natančnosti pa uporabimo naslednja pravila:

- $\delta(x, (x, t)) = 0$
- $\delta(>>, (\tau, t)) = 0$
- $\delta(>>, (x, t)) > 0$

Se pravi, da nas zanimajo samo taki primeri, kjer v dnevniku ne moremo narediti premika in obenem v modelu ne obstaja na tistem mestu skrita akcija. Če si pogledamo rezultate za prejšnji primer, dobimo vrednosti

$\delta S(\gamma_1) = \delta S(\gamma_2) = \delta S(\gamma_3) = 0$, $\delta S(\gamma_4) = 1, \delta S(\gamma_5) = 1$ in $\delta S(\gamma_6) = 2$. Te rezultate lahko potem tudi normaliziramo, da 0 pomeni najmanjšo natančnost, 1 pa največjo natančnost. Poleg teh dveh metod obstajajo tudi druge metode za preverjanje skladnosti, kot je na primer preverjanje sledi, popravljanje modela, ... [18]

Poglavje 7

Druge perspektive procesa

Pri iskanju procesnega modela je glavni poudarek na control-flow vidiku. Poleg same informacije o poteku procesa pa lahko dnevniku dogodkov vsebujejo tudi ostale pomembne podatke, kot so na primer organizacijske strukture v podjetju, različne informacije v zvezi s primerom, ter pomembne lastnosti kar se tiče časa in trajanja izvedbe različnih delov procesa. Tako so poleg control-flow vidiku, zelo pomembne tudi organizacijski vidik, vidik s strani primera ter časovni vidik. Organizacijski vidik je pomemben zato, ker dobimo pogled v tipične delovne vzorce, organizacijske strukture in socialne mreže. Iz časovnega vidika in pogostosti pojavljanja zahtev lahko dobimo podatke za detekcijo ozkih grl ter ostalih zmogljivostnih problemov. Vidik s stani primera pa lahko uporabimo za boljše razumevanje odločanja in analizo razlik med različnimi primeri posameznih zapisov.[3]

7.1 Organizacijski vidik

Pri organizacijskem vidiku se osredotočamo na probleme glede na tri vrste procesnega rudarjenja, in sicer odkrivanje, skladnost in izboljšava. Pri iskanju nam je glavni cilj izgradnja modela, s katerim ponazorimo trenutno situacijo. Za organizacijski vidik sta pomembna predvsem dva modela. To sta organizacijski model, ki predstavlja trenutno organizacijsko strukturo in

socialno omrežje, katera prikaže komunikacijo v organizaciji.

Organizacijski model je po navadi sestavljen iz organizacijskih enot, vlog(zadolžitev), izdajatelji in njihova razmerja(kdo pripada kateri enoti, kdo ima kako vlogo, njihova hierarhija). Pri analizi dnevnika dogodkov je težko odkriti natančno hierarhijo organizacijskih enot, vendar pa je mogoče prikazati skupino udeležencev, kateri izvajajo podobne naloge, kajti le določene skupine udeležencev, ima možnost izvrševanje določenih nalog. Iz tega, kako pogosto določeni udeleženci izvedejo specifične naloge, lahko ustvarimo skupine s podobnimi lastnostmi. Socialna mreža pa je mreža, v kateri vozlišča predstavljajo posameznike ali organizacijske enote, povezave med njimi pa njihovo razmerje. Generirana socialna omrežja omogočajo spremljanje organiziranost ljudi in skupin, ki delajo skupaj. Socialna omrežja lahko analiziramo z ogromno skupino tehnik SNA(Social Network Analysis), katera izračunajo metrike, kot so centralnost, pozicijo in gostoto.

Poleg tega lahko uporabimo iskana pravila, kot so pravila za dodelitve osebju in pravila za dodelitev nalogodajalcu. Primer enega takšnih pravil je na primer, da je pri popravilu mobilnega telefona to delo dodeljeno inženirju, kateri pripada tej mobilni ekipi. Medtem ko pravila za dodelitev osebju definirajo katero opravilo lahko izvrši posameznik pa pravila za dodelitev nalogodajalcu definirajo komu je določena naloga dodeljena ob izvrševanju. Delo lahko razvrščamo glede na prioriteto dela, kapaciteto nalogodajalcev, FIFO/LIFO pravil in podobno.

Poleg iskanja, pri katerem nam je cilj izgradnja modela, pa lahko tudi preverimo skladnosti modela. Pri preverjanju skladnosti raziskujemo, če se modelirano obnašanje sklada z opazovanim obnašanjem okolja. Obstajata dve dimenziji preverjanja s stališče control flow: natančnost in ustreznost. Natančnost je stopnja ujemanja med zapisi v dnevniku dogodkov in izvedbo poti v procesnem modelu. Ustreznost pa je stopnja natančnosti, s katero procesni model opiše opazovano obnašanje. Te koncepte lahko uporabimo tudi v organizacijskem vidiku. Na primer za pravila dodelitve nalog osebju; lahko priredimo natančnost kot razširitev, s se katero dejanski nalogodajalci

v dnevniku dogodkov skladajo z nalogami, katere so določene s pravili za dodeljevanje osebja. Ustreznost pa je kot stopnja natančnosti, s katero pravila za dodeljevanju osebja opišejo opazovano obnašanje.

Zadnja vrsta mogoče analize je izboljšava, katere cilj je obogatiti obstoječi model z razširitvijo modela glede na informacije, pridobljene iz dnevnika dogodkov. Kot primer tega lahko vzamemo razširitev socialnih omrežij z uporabo performančnih podatkov. Z identifikacijo ozkih grl ter njihovo projekcijo na socialno mrežo lahko identificiramo probleme v komunikacijski in organizacijski strukturi.[19]

7.1.1 Rudarjenje organizacijskega modela

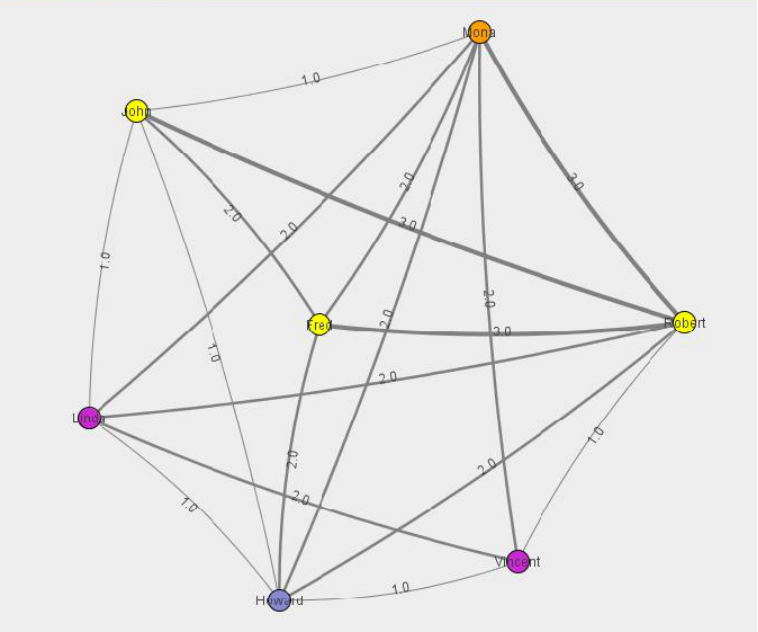
Rudarjenje organizacijskega modela cilja na pridobitev organizacijskega modela preko dnevnika dogodkov. Ker ima dnevnik dogodkov omejene informacije, ki so po navadi pomembne le za izvajanje procesov(izvajajoče naloge, izvajalci,..), ne moremo pridobiti dejanskega organizacijskega modela organizacije, vendar pa lahko izvlečemo skupino izvršiteljev, kateri imajo podobne karakteristike v izvajanju procesov in razmerja med temi skupinami in nalogami, ki jih izvajajo. Obstajata dva tipa organizacijskih subjektov, katera lahko izvlečemo iz dnevnikov dogodkov: opravilno temelječe skupine in primer temelječe skupine. Opravilno temelječe skupine so sestavljene iz ljudi, ki imajo pravico izvajati podobna opravila oz. naloge, medtem ko na primeru temelječe skupine vsebujejo ljudi, ki opravljajo z enakimi primeri. Opravilno temelječe skupine so pomembne za funkcionalno razdelitev na oddelke, v katerih imajo zaposleni podobne sposobnosti in znanje za opravljanja podobnih nalog. Na primeru temelječe skupine pa so pomembne za določitev projektnih skupin, v katerih imajo zaposleni različne sposobnosti, vendar pa delajo skupaj na enakem primeru. Čeprav ima veliko organizacij funkcionalne strukture, pa imajo nekatere kot so bolnišnice, svetovalne firme, .. potrebo da ustvarijo ekipe posameznikov z različnimi znanji in sposobnostmi, da dosežejo določen cilj(kirurške operacije, svetovalni projekti,...). V teh primerih je identifikacija teh skupin uporabna za razumevanje delova-

nja organizacije.[19]

7.1.2 Analiza socialnih omrežij

Za pridobitev socialnih mrež iz dnevnikov dogodkov je bilo razvitih precej metrik. Za boljše razumevanje si je najbolje ogledati osnovni koncept. Osnovna ideja je nadzirati, kako posamezni primeri potekajo med izvršitelji nalog. Tipični primer je metrika podajanja dela. Če obstajata dve vzorčno povezani aktivnosti v enem primeru, katerega prvo aktivnost je dokončal izvršitelj i in drugo izvršitelj j, je zelo velika verjetnost, da je bilo med njima podano delo od izvršitelja i do izvršitelja j. Zato lahko povežemo vozlišče i in vozlišče j. Ta način prepoznavanja lahko osvežimo na različne načine. Na primer, lahko uporabimo znanje strukture procesa, če dejansko obstaja odvisnost med dvema aktivnostima. Uporabimo pa lahko ne samo dejansko uspešno odkrito povezavo, ampak tudi ne direktno uspešnost z uporabo »causality fall faktorja«. Če imamo n aktivnosti med aktivnostima, ki sta ju končala izvršitelj i in izvršitelj j, je causality fall faktor B_n. Še en primer je subcontracting metrika, kjer je glavna ideja štetje števila izvedb aktivnosti izvršitelja j med izvedbo aktivnosti izvršitelja i. To lahko nakazuje predajanje dela od i do j. Z uporabo teh in podobnih metrik lahko iz podatkov pridobljenih iz dnevnikov dogodkov naredimo socialno mrežo.

Po generiranju socialnih mrež lahko uporabimo različne SNA tehnike, kot so: gostota, osrednjost, kohezija, enakost in druge. Na premier z uporabo betweenness(razmerje temelječe na število obiskanih geodetskih poti določenega vozlišča) lahko ugotovimo možna ozka grla. Če pa uporabimo metriko predaje dela, lahko ugotovimo izvršitelje, ki samo sprožijo proces, saj nimajo prihajajočih povezav ter tiste, ki delajo samo na končnih aktivnostih, saj nimajo odhajajočih povezav. Nato, če uporabimo subcontracting metriko, pa začetno vozlišče predstavlja izvršitelja in končno vozlišče podizvršitelja. Zato so tista vozlišča z visoko vrednostjo odhajajoče centralnosti izvršitelji del, medtem ko tisti z visoko vrednostjo prihajajoče centralnosti podizvršitelji. Če pa uporabimo joint case metriko pa vidimo, da tisti z vi-



Slika 7.1: Primer socialne mreže ustvarjene s programom Prom [33]

soko gostoto bolj sodelujejo med sabo. Temu se pravi tudi raven ego omrežja, pri katerem vidimo koliko izvršitelji sodelujejo med samo. V praktičnih primerih se je pokazalo, da z uporabo teh tehnik in metod na realnih dnevnikih dogodkov dobimo dosti bolj natančno sliko sodelovanja in komunikacije med zaposlenimi, kot pa, da bi uporabili na primer, metode kot so analiza pošiljanja e-mailov med njimi. Saj je tukaj dosti bolj povezano s primeri samo iz delovnega sodelovanja in ni »onesnaženo« z nedelovnimi primeri(kot so razni maili, iz zasebnih razlogov, ipd.). [19]

7.2 Časovni vidik

Časovni vidik procesnega rudarjenja je namenjen analizi časov in pogostosti pojavljanja dogodkov. Večina dnevnikov dogodkov vsebuje časovni žig, vendar pa se razlikujejo po podrobnosti podanih časovnih parametrov. Nekateri imajo samo datum, drugi pa časovni žig do milisekunde natančnosti dogodka.

Kot cilj časovnega vidika in s tem možnost pri prisotnosti časovnih žigov v dnevnikih dogodkov so iskanje ozkih grl, analiza ravni storitve, nadzor uporabe resursov in napoved trajanja procesov za dokončanje trenutno tekočih primerov.

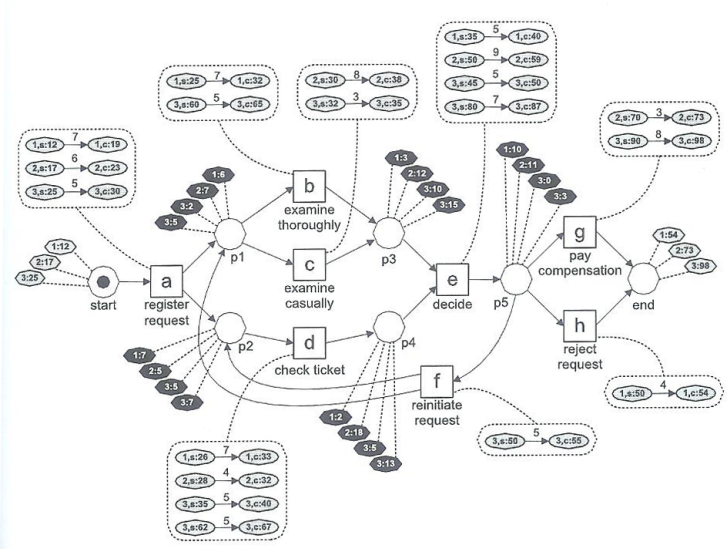
V naslednjem primeru si lahko pogledamo, kako se začne in kakšne so osnovne analize časovnega vidika. Vzemimo kot primer naslednji zapis v dnevniku dogodkov, kot je prikazan v tabeli 7.1.

Cace ID	Trace
1	$\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}, b^{32}_{complete}, d^{33}_{complete}, e^{35}_{start}, e^{40}_{complete}, h^{50}_{start}, h^{54}_{complete}\rangle$
2	$\langle a^{17}_{start}, a^{23}_{complete}, d^{28}_{start}, c^{30}_{start}, d^{32}_{complete}, c^{38}_{complete}, e^{50}_{start}, e^{59}_{complete}, g^{70}_{start}, g^{73}_{complete}\rangle$
3	$\langle a^{25}_{start}, a^{30}_{complete}, c^{32}_{start}, c^{35}_{complete}, d^{35}_{start}, d^{40}_{complete}, e^{45}_{start}, e^{50}_{complete}, f^{50}_{start}, f^{55}_{complete}, b^{60}_{start}, d^{62}_{start}, b^{65}_{complete}, d^{67}_{complete}, e^{80}_{start}, e^{87}_{complete}, g^{90}_{start}, g^{98}_{complete}\rangle$
...	...

Tabela 7.1: Primer predstavitve časovnih parametrov. [3]

V tej tabeli vidimo, da imamo podane časovne parametre kot dvoštevilčne zapise namesto dejanskih datumov. To je zaradi lažjega predstavljanja, v realnih primerih imamo podan datum kot časovni žig. Iz teh podatkov smo dobili časovno analizo grafično predstavljeno na sliki 7.2 .

Na tej sliki vidimo časovne izračune za vsako aktivnost in vsak prehod posebej. Če pogledamo za aktivnost a vidimo, da imamo tri instance aktivnosti. Prva instanca se je začela v a s časovnim žigom 12 in se končala s časovnim žigom 19. Vse skupaj pa je trajalo 7 časovnih enot. Potem je v pogoju p_1 instanca 1 trajala 6 časovnih enot in se potem nadaljevala v aktivnosti b. Te časovne parametre izračunamo za celotni procesni model. Moramo pa ignorirati tiste primere, kateri se ne dokončajo in kateri nimajo 100% skladnosti z modelom. Že s pogledom in analizo takega modela s temi dodanimi podatki, lahko dobimo kake pomembne podatke. Na primer, v tem našem modelu vidimo, da je za primer tretje instanca aktivnosti a potekala od časa 25 do 30. Pri času 30 sta se aktivirala c in d. Vendar pa se je c komaj



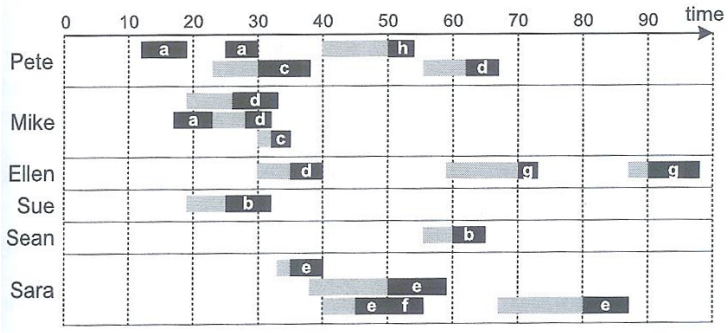
Slika 7.2: Primer izračun časovnih aktivnosti za vsak prehod. [3]

začel pri času 32 in d pri času 35. Tako vidimo, da ima c čakalno dobo 2 in d čakalno dobo 5 časovnih enot.

Po simulaciji vsakega primera skozi model dobimo multi-set trajanj za vsak pogoj in akcijo. Primer za p_1 je multi-set trajanj $[6, 7, 2, 5, \ldots]$. Za velike sisteme imamo zelo veliko teh števil in lahko izračunamo standardno deviacijo, povprečje, maksimum in minimum. Prav tako pa lahko izračunamo interval zaupanja. Primer »90% zaupanja vrednega intervala za povprečje čakanja za aktivnost x je med 40 in 50 minutami«. Poleg vsega naštetega je možno izvleči še naslednje informacije:

- Vizualizacija časa serviranja in čakanja – Statistike, kot so povprečno čakanje na aktivnost, lahko projektiramo na procesni model. Aktivnosti z zelo visokimi variacijami v servisiranju primerov lahko označimo in podobno.

- Odkritje in analiza ozkih grl – Že prej ugotovljen multi-set lahko uporabimo za odkritje in analizo ozkih grl. Mesta, kjer se porabi največ časa, se lahko označijo ter potem analizirajo zakaj se tam porabi največ



Slika 7.3: Instance aktivnosti projicirane na resurs. [3]

časa.

- Čas trajanja in SLA(Service Level Agreements) analiza – čas trajanja izračunamo glede na multi- set trajanj, lahko izračunamo povprečen med x in y ali del primerov, ki traja več kot ta čas in podobno. To lahko uporabimo za preverjanje SLA. Lahko je na primer v pogodbi zapisano, da 90% primerov mora biti končanih v določenem roku in tiste, ki temu ne ustrezajo, označimo.

- Analiza pogostosti in izkoriščenosti – ko smo izvajali primere zapisane v dnevnike dogodkov, lahko tudi zabeležimo pogostost izvajanja določenih aktivnosti. Na primer po aktivnosti e imamo na razpolago aktivnosti f, g in h. Z analizo pogostosti lahko na primer vidimo, da ima f pogostost pojavljanja npr. 50%, g 35% in h 25%, ter nato z združitvijo pogostosti in povprečnega časa servisiranja, lahko dobimo izkoriščenost resursa.

V predstavljenem primeru ta vsebuje le start in complete dogodka. Lahko pa vsebuje tudi ostale, kot so assign, schedule, suspend, resume, ... V takem primeru lahko izvlečemo še dodatne statistike in analize. Na primer, če aktivnosti za assign sledi start, lahko analiziramo koliko časa rabi od trenutka ko se mu dodeli delo do dejanskega začetka dela.[3]

7.3 Vidik s strani primera

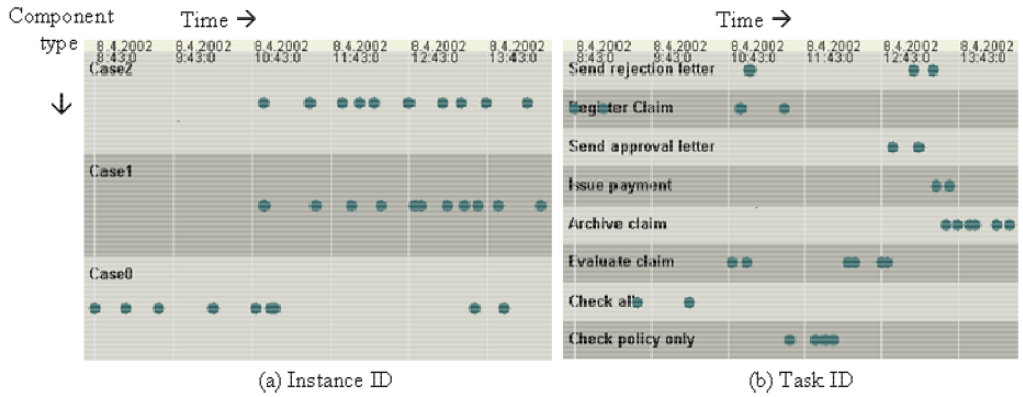
V vidiku s strani primera se postavlja vprašanje »kaj«. To analizo lahko izvedemo takrat, ko imamo podane različne lastnosti individualnih primerov. Nekatere lastnosti nam lahko podajo dodatno informacijo o aktivnostih, ki se izvajajo ali izvajalcih te aktivnosti, lahko pa nam podajo še kake informacije o primeru. Kakor koli je tesno povezano praktično izvajanje primera in lastnosti povezane s primerom. V vidiku s strani primera se ukvarjamo ravno s takšnimi povezavami. Želimo ugotoviti različna pravila s katerimi lahko pojasnimo obnašanje v določenih aktivnostih. Se pravi neke vrste if ... then ... pravila. Ter poiskati primere v katerih se ta pravila ne ujemajo in poiskati vzroke za to.[20]

7.4 Prikaz dogodkov na prvi pogled(Točkast grafikon)

Analizo s točkastim grafikonom lahko uvrstimo med iskalne tehnike, vendar pa se za razliko od obstoječih ta usmerja v bolj časovno dimenzijo in ne zahteva nobene posebne strukture procesa. Prikaže lahko posebno uporaben pogled, s katerim lahko hitro ugotovimo zmogljivostne probleme v procesu.

7.4.1 Pregled

Točkast grafikon je podoben Ganntovemu diagramu. Prikaže razpršenost dogodkov skozi čas. Osnovna ideja je postaviti točke glede na čas, kdaj so se zgodili. Če si pogledamo na primeru iz slike 7.4, vidimo, da točka na grafu predstavlja posamezni dogodek v dnevniku dogodkov. Graf ima dve dimenziji, na x osi označimo čas, na y osi pa tip komponente. Ta tip je lahko instanca, izvajalci, zadolžitev, ali podatkovni elementi. Na sliki 7.4 a.) vidimo, da je za komponento izbrano opravilo. Na sliki b.) pa je kot komponenta izbrana instanca procesa. S to izbiro se uporabniku pokaže katere instance potekajo dlje, katere imajo polno dogodkov in podobno. V

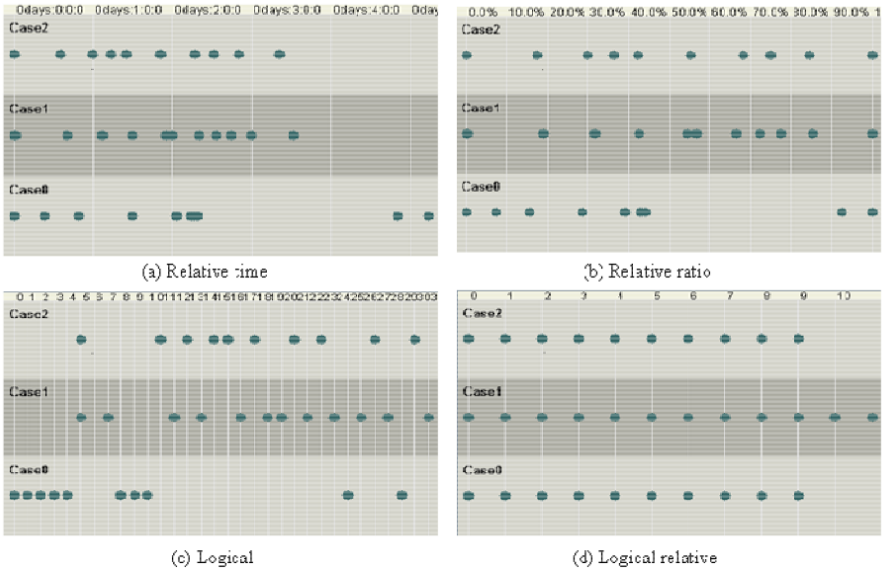


Slika 7.4: Primer točkastega grafa. [21]

grafu imamo možnost izbire različne prikaza časa in nekaj metrik za prikaz zmogljivosti.

7.4.2 Časovne možnosti

Analizo s točkastim grafikonom nam omogoča izbiro več časovnih prikazov. Prva možnost je dejanski prikaz časovnih žigov, torej nam prikaže, kdaj se je dogodek zgodil glede na realni čas. Kot lahko vidimo prikazano na sliki 7.4. Obstajajo pa štiri alternative temu prikazu. Prva izmed teh je relativni prikaz časa. Pri tem prikazu je prvi dogodek vsake komponente postavljen na 0. V našem primeru na sliki 7.5 (a), vidimo, da je v tej instanci procesa »Case0« vzelo največ časa, čeprav je bil končan pred drugima dvema. Drugi prikaz je relativno razmerje, pri katerem raztegne vsak primer do konca časovnega intervala. Tu vidimo relativno porazdelitev dogodkov v vsaki instanci procesa, kar je prikazano na sliki 7.5 (b). Tretji način je logična možnost razvrstitve časa. Tukaj pa so dogodki razvrščeni glede na časovni zapis z dodano sekvenčno številko. To pomeni da ima prvi dogodek čas 0, drugi dogodek čas 1 in tako dalje(Slika 7.5 (c)). Nazadnje pa še logična relativna možnost, ki je pa združitev relativnega prikaza in logičnega prikaza, se pravi imamo prikaz, ki se začne z 0 (Slika 7.5 (d)).[21]



Slika 7.5: Različni prikazi točkastega grafa. [21]

7.4.3 Zmogljivostne metrike

Zaradi tega ker točkast grafikon prikazuje razpršitev dogodkov v dnevniku dogodkov, je težko pokazati tradicionalne zmogljivostne vrednosti, kot so čas čakanja in čas izvajanja, vendar pa je zato možno prikazati metrike za dogodke in njihovo razpršenost. Lahko uporabimo dve metriki za merjenje zmogljivosti. Prva je metrika za merjenje celotnega dnevnika dogodkov, druga pa za vsako komponento posebej. Za metriko celotnega dnevnika lahko izračunamo : pozicijo prvega in zadnjega dogodka v dnevniku, povprečno, minimalno in maksimalno razpršenost. Za vsako posamezno komponento pa pozicijo prvega in zadnjega dogodka v komponenti, povprečen, maksimalen in minimalen interval med dogodki. Glede na tip komponent in časovne opcije se lahko te rezultate interpretira na različne načine. Na primer če imamo izvajalca kot komponento in dejanski čas kot časovno opcijo, povprečna razpršenost predstavlja povprečen čas, ko je izvajalec dejansko zaposlen z nekim primerom. Če pa imamo instance za tip komponent in relativni čas kot časovna opcija, pa povprečna razpršenost predstavlja povprečen čas

izvajanja instanc. [21]

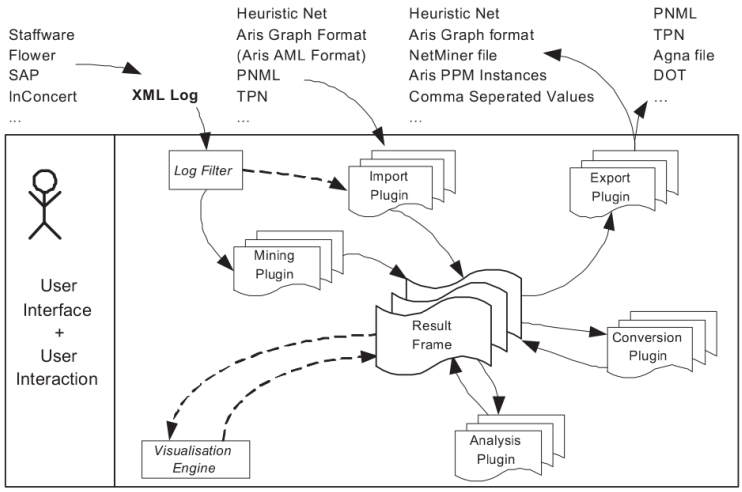
Poglavje 8

ProM

ProM(Process Mining framework)[34] je odprtokodno orodje za uporabo algoritmov, namenjenih procesnemu rudarjenju. ProM omogoča uporabnikom in razvijalcem platformo za enostavno uporabo in razširitev algoritmov za procesno rudarjenje. Glavni cilj razvijalcev tega orodja je postati de facto standard platforma za procesno rudarjenje v akademskem okolju. To pa želi doseči z vzpostavitvijo aktivne prepoznavne skupnosti uporabnikov in razvijalcev ter ustvariti zavedanje moči tehnologije procesnega rudarjenja. [22]

8.1 Arhitektura

Osnova za vsako procesno rudarjenje je dnevnik dogodkov. Ker ima vsak informacijski sistem večinoma svoj format za shranjevanje dogodkov, je bil za potrebe ProM-a razvit poseben splošni XML format. Ta format je bil osnovan na temeljiti primerjavi med vhodnimi podatki, ki jih rabimo za procesno rudarjenje in dejanskimi podatki, ki so že bili zapisani v dnevnikih dogodkih kompleksnih informacijskih sistemih.(ERP ali WFM). Druga pomembna funkcija ProMa je, da omogoča medsebojno povezovanje različnih vtičnikov. Vtičnik je v osnovi implementacija nekega algoritma, ki se ga uporablja za neko dejavnost v procesnem rudarjenju. Take vtičnike se brez težav dodaja v ogrodje in ni potrebe po ponovnem spreminjanju ogrodja Proma, dodamo



Slika 8.1: Arhitektura ProM-a [23]

samo ime v nekaj ini-datotek.

Na sliki 8.1 si lahko ogledamo razmerje med ogrodjem, dnevniki in vtičniki. Kot prikazuje slika, lahko ProM bere datoteke v XML formatu preko filtra komponent za dnevnike dogodkov. Ta komponenta je zmožna dela z velikimi podatkovnimi seti in jih filtrirati, preden se procesno rudarjenje dejansko začne. Preko import vtičnikov je možno naložiti ogromno modelov, vse od Petri mrež do LTL formul. Iskalni algoritmi opravijo rudarjenje in rezultat shranijo kot okvir. Te okvire lahko potem uporabimo za vizualizacijo, prikazovanje petri mrež, EPC, socialna omrežja, ali za nadaljnjo analizo in pretvorbo. Vtičniki za analizo pa vzamejo rezultate in jih analizirajo. Potem imamo še algoritme za pretvarjanje, ki rezultat procesnega rudarjenja pretvorijo v drug format.[23]

8.2 Razvoj in izboljšave v ProM6

V začetnih verzijah ProMa so implementirani algoritmi z vtičniki predpostavljali prisotnost grafičnega vmesnika. Večina algoritmov je rabila neke parametre in vtičnik je te parametre zahteval preko nekakšnega grafičnega

vmesnika, zato je bila zelo povezana uporaba vtičnikov z uporabo grafičnega vmesnika. To je pa imelo slabo stran, saj je bilo mogoče poganjati vtičnike samo na GUI-zavedajočem strežniku. Prav tako je bilo nemogoče učinkovito poganjati eksperimente v zvezi s procesnim rudarjenjem z uporabo razpršene infrastrukture in serijskega preverjanja sistemov. V verziji ProM 6 pa je bil ta problem odpravljen s previdno ločitvijo grafičnega vmesnika in vtičnikov.

Naslednji problem, ki je bil v starejših verzijah ProM-a je, da ogrodje ni vedelo, katere so vhodne zahteve in izhodna pričakovanja vtičnikov. Kot posledica tega pa ni bilo mogoče združevati več vtičnikov skupaj. Ta problem je bil prav tako rešen z izidom ProM 6. Prav tako so rešili problem z licenciranjem določenih paketov in posodobitvijo Package Managerja za namestitev vtičnikov.[24]

8.3 Ostala možna orodja

Poleg ProM-a obstaja tudi nekaj drugih komercialnih rešitev za procesno rudarjenje. Naslednjih nekaj opisanih orodij ima skupno to, da je z njimi možno samodejno, na podlagi dnevnikov dogodkov, odkriti model procesa.

8.3.1 Software AG - ARIS Process Performance Manager (PPM)

ARIS Process Performance Manager (PPM) pripada podjetju Software AG in je komponenta ARIS Controlling Platformi, usmerjena v analizo poslovnih procesov, glede na zgodovino zapisanih podatkov. Kot vhod za analizo to orodje omogoča različne tipe formatov od CSV datotek do dnenvikov dogodkov izvlečenih iz podatkovnih baz in zunanjih sistemov, kot so ERP-ji in CRM-ji. Odkrite modele lahko izvozimo kot XML, AML, image datoteka, podatke pa lahko zapišemo kot datoteko Microsoft Excel. Procesni model pa je mogoče odkrivati na dva načina.[25]

8.3.2 Fourspark – Flow

Flow je orodje razvito s strani Norveškega podjetja Fourspark. Podprti format kot vhod je datoteka formata Microsoft Excel. Uporabniški vmesnik je zgrajen kot nadzorna plošča in omogoča fleksibilnost s tem, ko da uporabniku možnost dodajanje različnih widgetov za prikaz različnih informacij. Poleg iskanja procesov omogoča še vizualizacijo modelov zgrajenih med dvema časovnima zapisoma. Redke primere je mogoče izključiti iz modela z nastavitvijo mejne vrednosti. Omogoča pa tudi vizualizacijo pogostejših poti z uporabo različnih barv in debelino črt.[25]

8.3.3 Futura Process Intelligence - Futura Reflect

Futura reflect je internetno bazirano orodje, ki lahko deluje kot samostojno orodje ali pa kot del BPM. Kot vhod podpira samo CVS format datotek. Rezultat pa lahko shranimo kot image datoteko ali kot FLOW model, ki ga lahko pozneje naložimo v BPM platformo. Funkcionalnost tega orodja vsebuje pregled nad vnesenimi podatkov, iskanjem procesnih modelov(uporablja genski algoritem), pregled in analizo končanih iskanj.[25]

8.3.4 QPR – ProcessAnalyzer

To orodje bazira na programu Microsoft Excel in se namesti kot Add-on. Omogoča uvoz katerekoli podatke, ki so že v Excelu. Omogoča iskanje procesov, iskanje variacij v procesih in različne analize podatkov.[25]

Poglavje 9

Analiza realnega primera iz okolja

Za analizo realnega primera iz okolja sem izbral podatke, ki so dostopni na internetu in sicer na spletni strani [35]. Podatki so pridobljeni z beleženjem procesov v dejanskem informacijskem sistemu za upravljanje s prometnimi prekrški. Podatki predstavljajo en proces od začetka, ko se sproži zahteva za izdajo kazni pa vse do plačila oz. v primeru, da je pritožba uspešna do obvestitve o oprostitvi plačila. Na tem primeru bom poizkušal izvesti večino predstavljenih tehnik za odkrivanje in izboljšavo procesov, obenem pa poskušal odgovoriti na zastavljena vprašanja v naslednjem odstavku.

9.1 Cilji in namen analize

Namen analize je prikazati tehnike procesnega rudarjenja na realnem primeru. Cilj analize pa je odgovoriti na naslednja zastavljena vprašanja v zvezi s procesom:

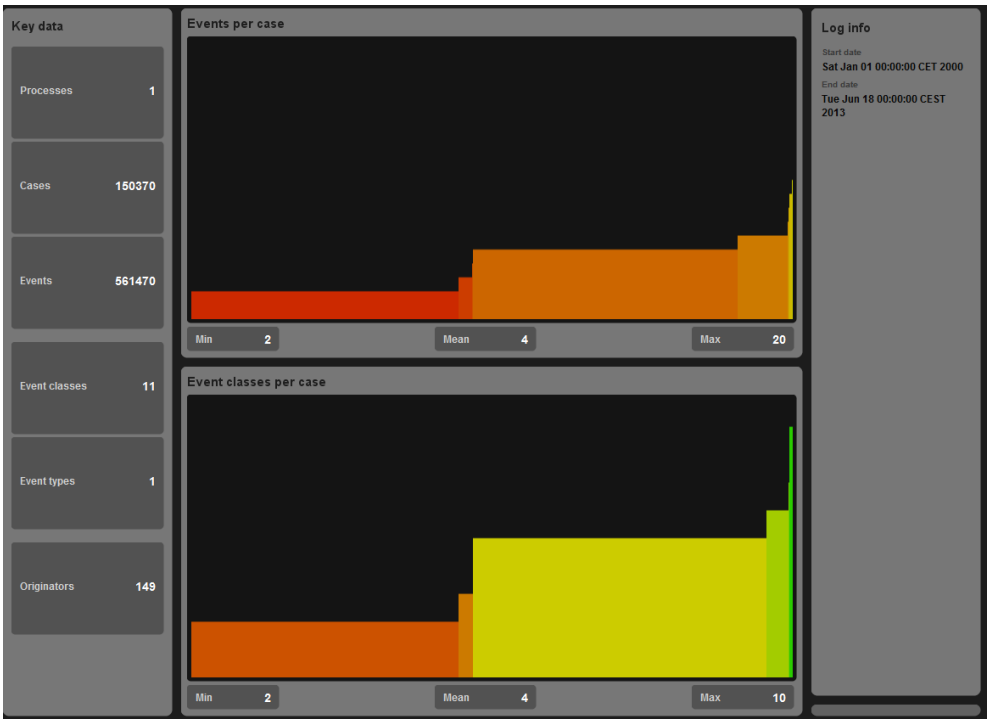
1. Koliko je zapisov v dnevniku dogodkov in koliko je dokončanih primerov v njem?

2. Kakšne so lastnosti zapisanih podatkov dnevniku dogodkov?

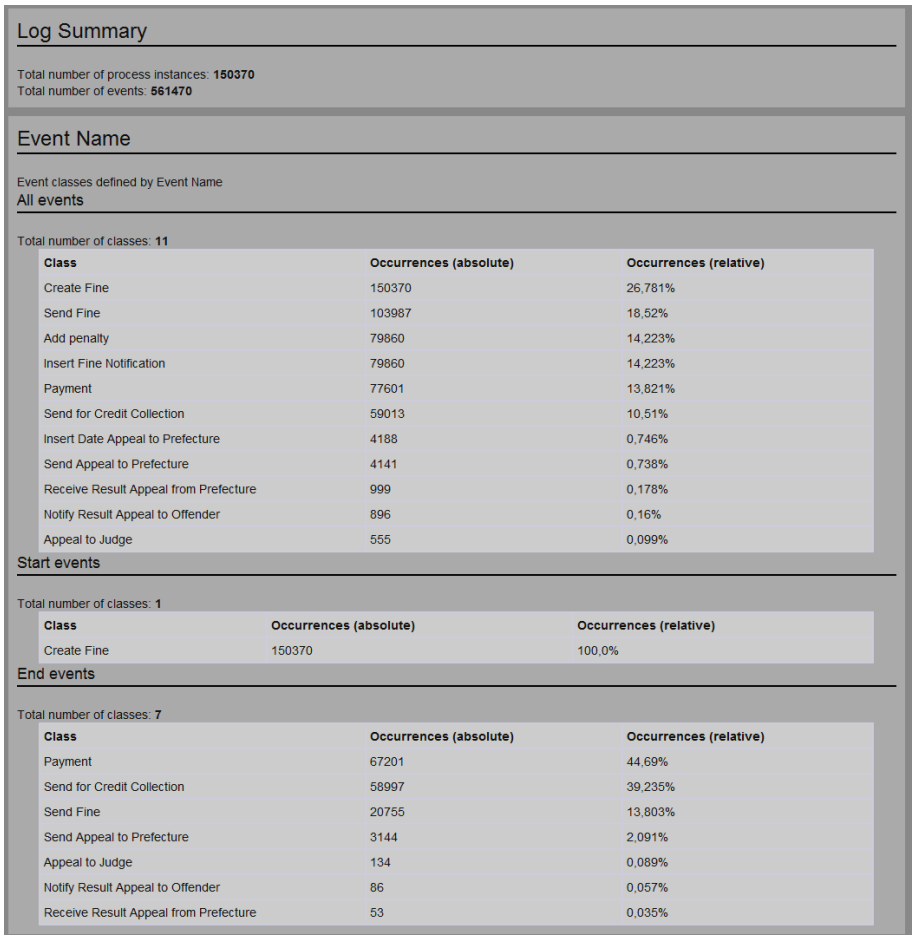
3. Ali je proces strukturiran?
4. Koliko časa traja najdlje izvajajoči primer v procesu in koliko je povprečje trajanja enega primera?
5. Kako so aktivnosti med sabo povezane?
6. Ali je mogoče ustvariti procesni model za izbrani proces ter kako natančen je ta model?
7. Kako so izvrševalci aktivnosti v procesu med seboj povezani?
8. Katere aktivnosti trajajo najdlje v procesu oz. kje so ozka grla v procesu?

9.2 Pregled podatkov

Prvo, kar lahko s programom ProM naredimo je, da pregledamo lastnosti podatkov. Na sliki 9.1 lahko razberemo lastnosti podatkov, kot jih je predstavil program ProM. Najprej vidimo, da imamo en proces, kateri vsebuje 150370 primerov, ter 561470 dogodkov. Kot primer je v tem procesu en kaznjenec z zahtevkom za plačilom. Dogodek v tem primeru pa je vsak pojav, ki se zgodi od ustvaritve kazni do plačila oz. pritožbe. Iz tega pregleda podatkov lahko prav tako razberemo, da so se podatki začeli beležiti 1.januarja 2000 in končali 18.junija 2013. Razberemo pa tudi, da je v tem informacijskem sistemu sodelovalo 149 izvršiteljev procesov. Ti izvršitelji so lahko sistem sam ali pa zaposleni v tem uradu. Vsega skupaj imamo 11 različnih možnih dogodkov, ki se lahko zgodijo v tem informacijskem sistemu. V povprečju se zgodijo 4, maksimalno pa vseh 11. Medtem pa se v samem procesu lahko večkrat ponovijo različni dogodki, kar nam prikazuje prvi graf na tej sliki, kjer je prav tako povprečje dogodkov na primer 4, medtem ko pa je maksimalno število dogodkov, ki se zgodijo na primeru, enako 20. Naslednji prikaz, ki ga omogoča ProM je povzetek vseh podatkov v dnevniku dogodkov. To vidimo na sliki 9.2. Pri tem povzetku vidimo, kolikokrat se zgodi določeno



Slika 9.1: Prikaz skupek lastnosti podatkov v ProMu



Log Summary

Total number of process instances: **150370**
Total number of events: **561470**

Event Name

Event classes defined by Event Name

All events

Total number of classes: **11**

Class	Occurrences (absolute)	Occurrences (relative)
Create Fine	150370	26,781%
Send Fine	103987	18,52%
Add penalty	79860	14,223%
Insert Fine Notification	79860	14,223%
Payment	77601	13,821%
Send for Credit Collection	59013	10,51%
Insert Date Appeal to Prefecture	4188	0,746%
Send Appeal to Prefecture	4141	0,738%
Receive Result Appeal from Prefecture	999	0,178%
Notify Result Appeal to Offender	896	0,16%
Appeal to Judge	555	0,099%

Start events

Total number of classes: **1**

Class	Occurrences (absolute)	Occurrences (relative)
Create Fine	150370	100,0%

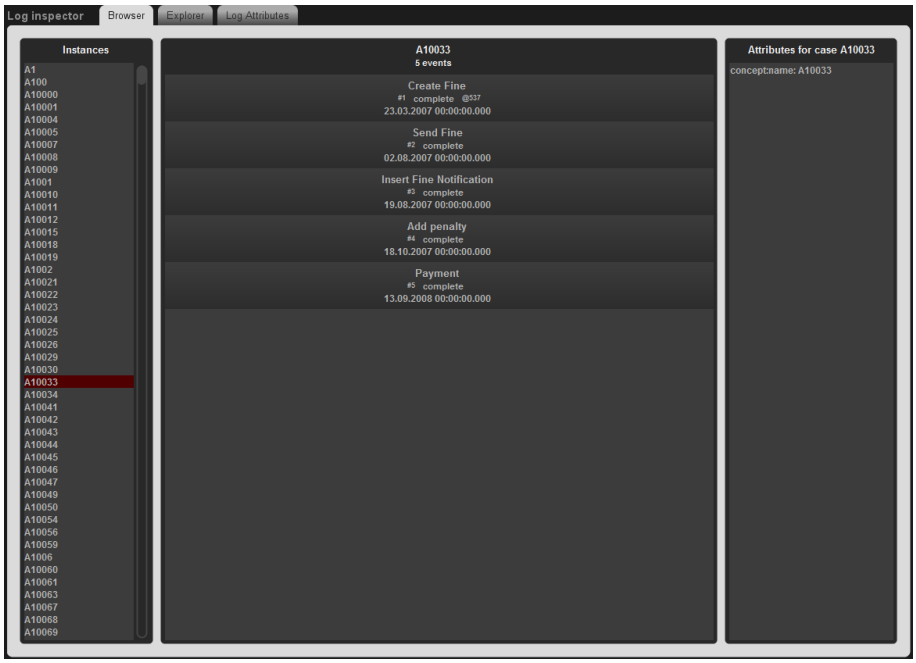
End events

Total number of classes: **7**

Class	Occurrences (absolute)	Occurrences (relative)
Payment	67201	44,69%
Send for Credit Collection	58997	39,235%
Send Fine	20755	13,803%
Send Appeal to Prefecture	3144	2,091%
Appeal to Judge	134	0,089%
Notify Result Appeal to Offender	86	0,057%
Receive Result Appeal from Prefecture	53	0,035%

Slika 9.2: Prikaz lastnosti podatkov v ProMu

opravilo ter kakšen je odstotek tega opravila v dnevniku dogodkov. Prav tako vidimo vsa začetna in končna vozlišča, ter glede na te podatke vidimo, da ta dnevnik dogodkov vsebuje tudi nedokončane oz. nepopolne dogodke. To vidimo iz končnih vozlišč, saj se v nekaterih primerih zaključijo brez da bi bilo to končno vozlišče. Zaključeni primeri se končajo na naslednje tri možne načine. Ali z dogodkom »Payment« ali »Send for credit collection« ali pa z »Notify result apeal to Offender«. Zato bom moral te podatke odstraniti iz dnevnika dogodkov, saj bodo v nasprotnem primeru pri analizi samo v napoto in bodo povzročali probleme pri rezultatih.



Slika 9.3: Prikaz podatkov v ProMu

Možen pa je še prikaz dejanskih podatkov po instancah, pri kateri je ena instanca ena posamezna sled v dnevniku dogodkov. To je prikazano na sliki 9.3. Kot vidimo, so bili podatki verjetno skopirani iz več tabel ter združeni v tej xes datoteki. Kot posledica tega je, da si id instance ne sledijo zaporedoma po datumu in času kdaj so se dejansko zgodile. To pa bo predstavljalo težave pri predstavitvi podatkov s točkastim diagramom. Zato bom moral pred začetkom analize popraviti tudi to.

Odločil sem se, da bom najprej popravitl instance in pripadajoče datume. Za kaj takega sam program ProM nima nobenega vtičnika ali kakega podobnega orodja, zato sem spisal preprost java program za odpravo teh težav v dnevniku dogodkov. Tako sem instance oz. sledi procesa oštevilčil od 100000 dalje. Saj ProMu predstavljajo težave pri prikazu tudi razločevanje med večmestnimi števili(npr. med 1006 in 10060, če imamo prej sled 10059 potem sled 1006 vstavi med te dve).

Ko sem odpravil problem z instancami pa sem uporabil vtičnik v ProMu,

ki se imenuje Simple Heuristic Miner, s katerim je možno filtrirati podatke in izbrati, katere sledi s končnimi vozlišči naj obdrži. Prav tako je tudi možno izbrati, koliko odstotkov sledi naj upošteva kot relevantne. Moje nastavitve so bile, da naj obdrži sledi, ki se končajo z »Payment« ali »Send for credit collection« ali pa z »Notify result apeal to Offender«. Prav tako pa sem izbral, da se naj ohrani 100% število vseh sledi s temi pogoji, saj bi v nasprotnem primeru imel sledi samo za najpogostejše primere.

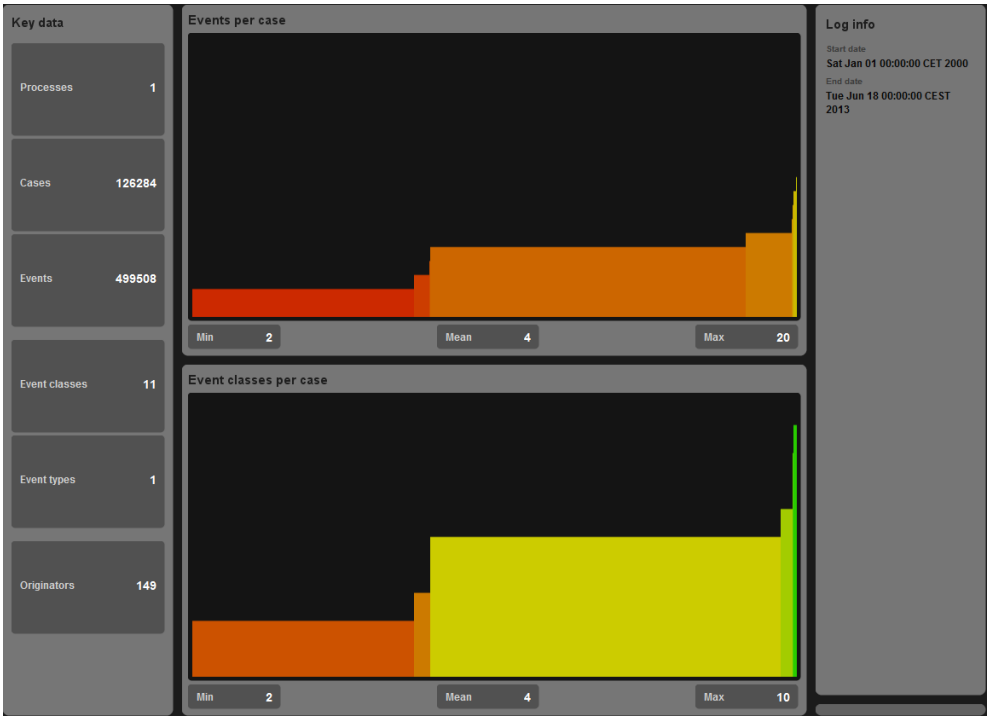
Na koncu tega filtriranja sem dobil dokončne podatke, ki jih lahko uporabim v analizi procesa. Pri pregledu vizualnega prikaza dokončnih podatkov v ProMu na sliki 9.4 vidimo, da se v povprečju izvedejo 4 dogodki na en primer. Največ dogodkov, ki se jih je zgodilo za en primer je 20. Na spodnji sliki pa vidimo, da se je v povprečju zgodilo prav tako 4 različni dogodki na en primer. Največ različnih dogodkov, ki se jih je zgodilo pa je 10. Število primerov, ki jih bom analiziral je 126284 število dogodkov, ki se jih je zgodilo pa je 499508.

Na sliki 9.5 si lahko ogledamo, kako so dejanske aktivnosti porazdeljene glede na odstotek pojavljanja. Prav tako pa lahko vidimo število končnih in začetnih vozlišč. Sedaj lahko vidimo, da se je število končnih vozlišč spremenilo na 3. Po pregledu porazdeljenih odstotkov glede na to, s katero aktivnostjo se proces konča vidimo, da se zelo malo primerov konča z dogodkom »Notify result apeal to Offender«. To nam pove, da je samo v zelo malem odstotku pritožb glede na kazen dejansko ugodeno.

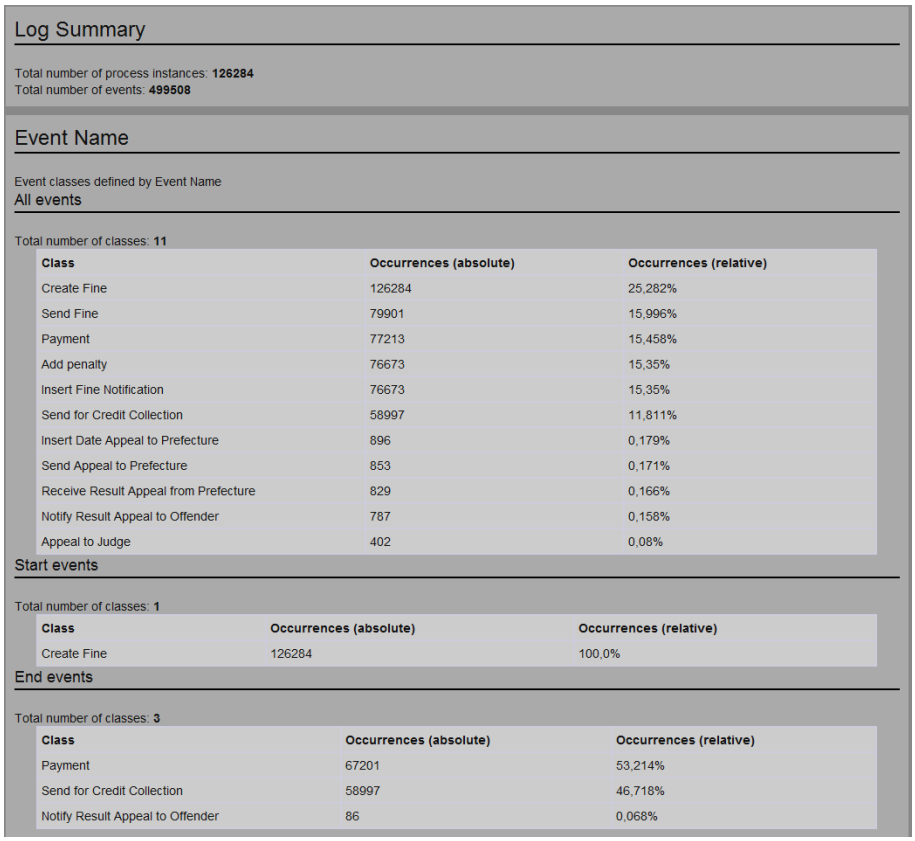
Na koncu si lahko na sliki 9.6 pogledamo, kako izgledajo dejansko zapisani podatki s spremenjenimi instancami in popravljenim vrstnim redom.

9.3 Analiza s točkastim diagramom

Ponavadi je najlažje začeti analizo procesa s točkastim diagramom, aj pri tem dobimo kar nekaj osnovnih podatkov glede samega procesa. Zato sem se tudi odločil, da začnem analizo s tem. Če kot tip komponente izberemo instanco in časovno komponento, dobimo izpis kot ga prikazuje slika 9.7. Ta grafikon nam



Slika 9.4: Prikaz lastnosti dokončnih podatkov v ProMu



Log Summary

Total number of process instances: **126284**
Total number of events: **499508**

Event Name

Event classes defined by Event Name

All events

Total number of classes: **11**

Class	Occurrences (absolute)	Occurrences (relative)
Create Fine	126284	25,282%
Send Fine	79901	15,996%
Payment	77213	15,458%
Add penalty	76673	15,35%
Insert Fine Notification	76673	15,35%
Send for Credit Collection	58997	11,811%
Insert Date Appeal to Prefecture	896	0,179%
Send Appeal to Prefecture	853	0,171%
Receive Result Appeal from Prefecture	829	0,166%
Notify Result Appeal to Offender	787	0,158%
Appeal to Judge	402	0,08%

Start events

Total number of classes: **1**

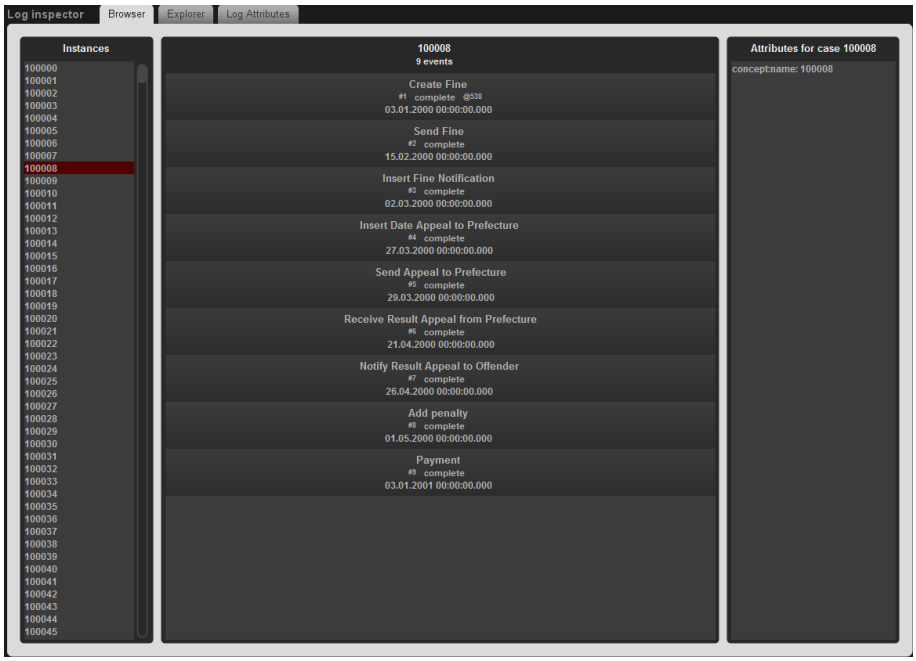
Class	Occurrences (absolute)	Occurrences (relative)
Create Fine	126284	100,0%

End events

Total number of classes: **3**

Class	Occurrences (absolute)	Occurrences (relative)
Payment	67201	53,214%
Send for Credit Collection	58997	46,718%
Notify Result Appeal to Offender	86	0,068%

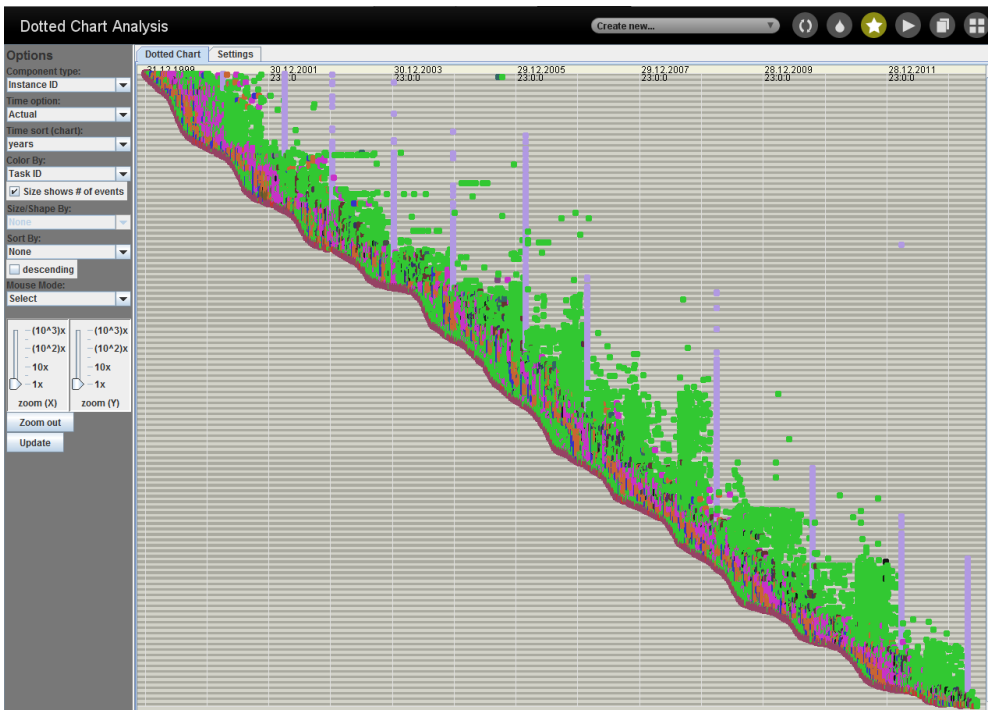
Slika 9.5: Prikaz skupka lastnosti dokončnih podatkov v ProMu



Slika 9.6: Prikaz dokončnih podatkov v ProMu

predstavlja, kako so se instance pojavljale skozi časovno obdobje. Na tem grafikonu lahko vidimo, da gre verjetno za strukturiran proces, saj zahteve večinoma enakomerno prihajajo v sistem in se večinoma enakomerno končajo s končnimi aktivnostmi. Barve, ki pripadajo določeni aktivnosti, vidimo na sliki 9.8. Vse instance procesa prihajajo relativno enakomerno v sistem in se relativno enakomerno zaključijo. Le tu pa tam vidimo, da se katera instanca zaključi pozneje, kot ostale v podobno prejetem časovnem intervalu. Opazimo tudi, da se aktivnost »Send for credit collection« zgodi samo ob določenih dnevih. Ker gre najverjetneje za strukturiran proces, bomo lahko uporabili vse prej opisane tehnike za procesno rudarjenje.

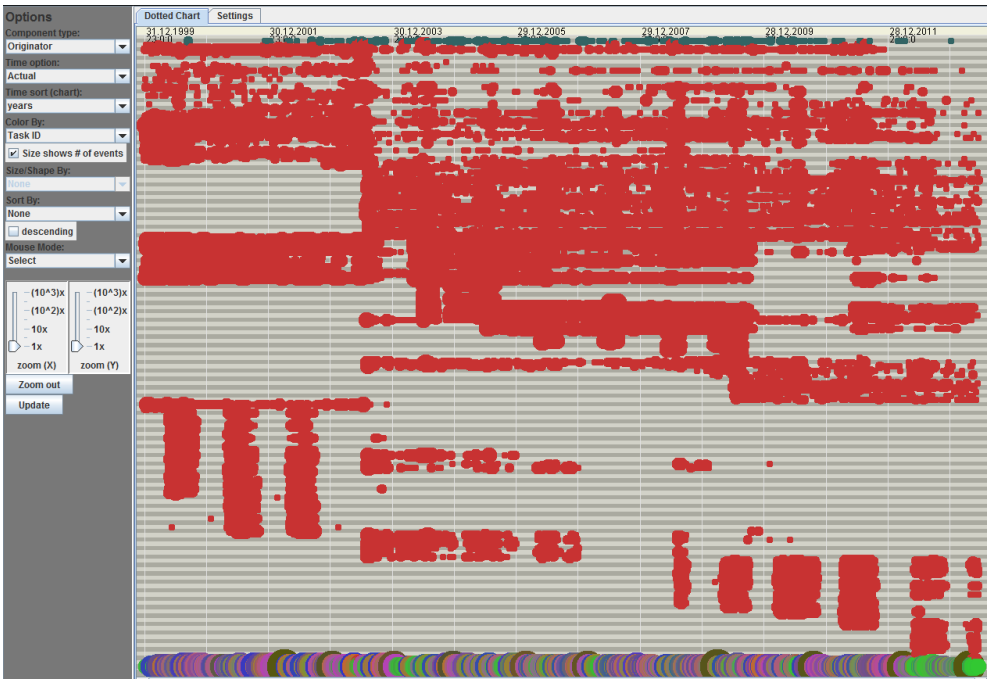
Naslednji prikaz v točkovnem diagramu, ki nam poda nekaj informacij o procesu je z izbiro tipe komponente za »originator«. S tem bomo dobili informacijo o izvrševalcih opravil na aktivnostih. Kot vidimo na sliki 9.9 so bili izvrševalci zabeleženi samo pri dveh opravilih za vse ostala opravila pa tega podatka ni. Zaradi tega ni mogoče tudi v nadaljevanju narediti



Slika 9.7: Analiza s točkastim diagramom

Add penalty:	push to change
Appeal to Judge:	push to change
Create Fine:	push to change
Insert Date Appeal t...	push to change
Insert Fine Notificati...	push to change
Notify Result Appeal ...	push to change
Payment:	push to change
Receive Result Appe...	push to change
Send Appeal to Pref...	push to change
Send Fine:	push to change
Send for Credit Colle...	push to change
<no name>:	push to change

Slika 9.8: Pripadajoče barve aktivnosti za določen proces.



Slika 9.9: Analiza s točkastim diagramom

in obravnavati kake posebne analize predaje dela. Ter kot lahko vidimo so vsi izvršitelji po tej analizi opravljali eno in isto delo. Se pravi izvršitelj aktivnosti »Create fine« je opravljal samo to opravilo, katera je označena z rdečo barvo. Možno je še spremeniti nekaj parametrov za kakšen drugačen prikaz grafikona. Vendar pa večino uporabnih stvari izvemo iz teh dveh grafikonov. Prav tako pa točkast diagram služi večinoma za hiter pregled procesa. Več informacij o procesu in delovanju izvemo iz naslednjih metod procesnega rudarjenja.

9.4 Procesni model

Večinoma je glavni cilj procesnega rudarjenja pridobiti dober model procesa iz dnevnika dogodka. Za odkrivanje procesnega modela obstaja veliko različnih algoritmov z veliko možnostjo prilagajanja. Za moj izbrani pro-

blem sem najprej pogledal kakšen rezultat vrne alfa algoritem. Vedel sem, da se v realnih primerih večinoma ne uporablja. Prav tako se je izkazalo, da ni ravno uporaben v mojem primeru. Proces je razdelil na več delov z različnimi vhodnimi vozlišči. Prav tako pa z algoritmom za skladnost ni mogel v popolnosti ponoviti nobene od sledi v dnevniku dogodkov. S pregledom obstoječih algoritmov za odkrivanje procesov sem ugotovil, da bi bil eden izmed možnih algoritmov hevrističen algoritem, saj imamo precej strukturiran proces, hevrističen algoritem pa bi naj ponavadi dajal precej dobre rezultate v takem primeru. Edini problem glede podatkov je bil še v tem, da so vsebovali več končnih vozlišč. V tem primeru se nepravilno izračuna natančnost algoritma, zato sem uporabil še en vtičnik v ProMu s katerim sem dodal končno vozlišče »ArtificialEnd«. Na sliki 9.10 si lahko pogledamo rezultat, ki ga vrne ta algoritem.

9.4.1 Hevristični algoritem

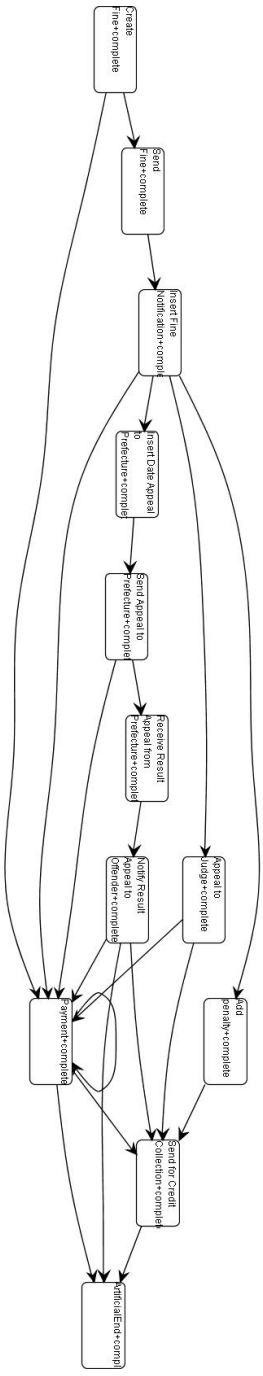
Kot vidimo na sliki 9.10 se proces začne z aktivnostjo »Create fine«, v tem procesu se ustvari kazen za storjen prekršek. Od tu sta možna dva scenarija, ali se kazen plača v določenem časovnem obdobju; v tem primeru se proces nadaljuje v »Payment« in zaključi. V nasprotnem primeru, če se pa kazen ne poravna v določenem obdobju, pa se proces nadaljuje v »Send Fine«, kjer se pošlje obvestilo , nato pa sledi proces »Insert Fine Notification«. Med tem časom ima kršitelj možnost pritožbe, kar bi se nadaljevalo v aktivnost »Insert Date Appeal to Prefecture«, medtem se sproži tudi aktivnost »Appeal to Judge«. Privzeti pregled procesa s hevurističnim algoritmom nam vrne rezultat brez pogleda, katere aktivnosti se zgodijo istočasno. Vendar pa ima tudi to možnost prikaza kar vidimo na sliki 9.11. Tam tudi vidimo, da se lahko takoj po aktivnosti »Appeal to Judge« zgodi aktivnost »Payment« (plačilo) ali aktivnost »Send for Credit Collection«, kjer pošljejo ljudi za terjatev kazni. Lahko pa se nadaljujejo v aktivnost »Send Appeal to Prefecture«. Ta se lahko zaključi z aktivnostjo »Payment«. Če ne, se nadaljuje z »Receive Result Appeal to Prefecture« kjer se prejmejo rezultati pritožbe. Vzporedno

se izvede še aktivnost »Add penalty«, v kateri se dodajo stroški ponovnega pošiljanja in še ostali možni stroški, ki so nastali v tem procesu. V primeru, da se proces zaključi z aktivnostjo »Notify Result Appeal to Offender« se teh stroškov in plačilu izogne in se proces tukaj konča. V nasprotnem primeru pa se še vse skupaj nadaljuje z »Payment« ali »Send for Credit Collection«.

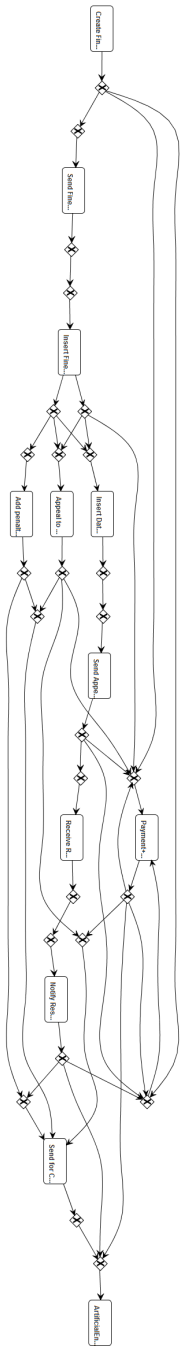
Za pregled povezav aktivnosti v procesu nam ta hevristični algoritem poda kar dobro sliko. Prav tako nam izpiše natančnost ujemanja procesa, kar je v mojem primeru 0,8708, to je že precej natančno. Saj velja nekakšno nenapisano pravilo, da kar je več kot 0,8 velja za strukturiran proces. To pomeni, da je proces možno kar dobro predstaviti in analizirati s procesnim rudarjenjem. Naslednji korak pri analizi procesa je pretvorba hevristične mreže(C-mreža) v petrijevo mrežo, saj je večino orodij za analizo možno opraviti le na petrijevi mreži. Pri tem pa nastanejo problemi, in sicer obstaja možnost pretvorbe v petrijevo mrežo, ampak pri tem nastanejo veliko število skritih akcij. Kljub poizkusu zmanjšanja teh skritih akcij z različnimi pristopi, jih vseeno nisem mogel dovolj zmanjšati. Nastale so pa tudi težave pri analizi petrijeve mreže. Program ProM je preprosto zamrznil pri poizkusu rekonstruiranja primerov in časovni analizi. Pri pregledu vseh možnosti, sem izbiro skrajšal na dva algoritma ali genski algoritem ali pa inductive miner. Kar se tiče algoritma bi morala oba zadoščati mojim zahtevam. Najprej sem poizkušal genski algoritem, vendar pa sem tudi po nekaj dneh poganjanja algoritma ostal brez petrijeve mreže. Zato sem za izgradnjo petrijeve mreže uporabil inductive miner.

9.4.2 Inductive miner

Pri izbiri inductive minerja sem dobil rezultat, kot ga prikazuje slika 9.12. Kot zahtevano natančnost sem izbral najmanj 0,9. Pridobljeno mrežo sem prav tako preveril z analizatorjem Woflan, kateri vrne rezultat, da je pridobljena mreža uglašena petrijeva mreža. Če si pogledamo sliko 9.12 vidimo, da se začne proces z začetno akcijo »Create Fine«. Ko se ta sproži, dobimo na



Slika 9.10: Hevristični model



Slika 9.11: Hevristični model s prikazom istočasnosti akcij

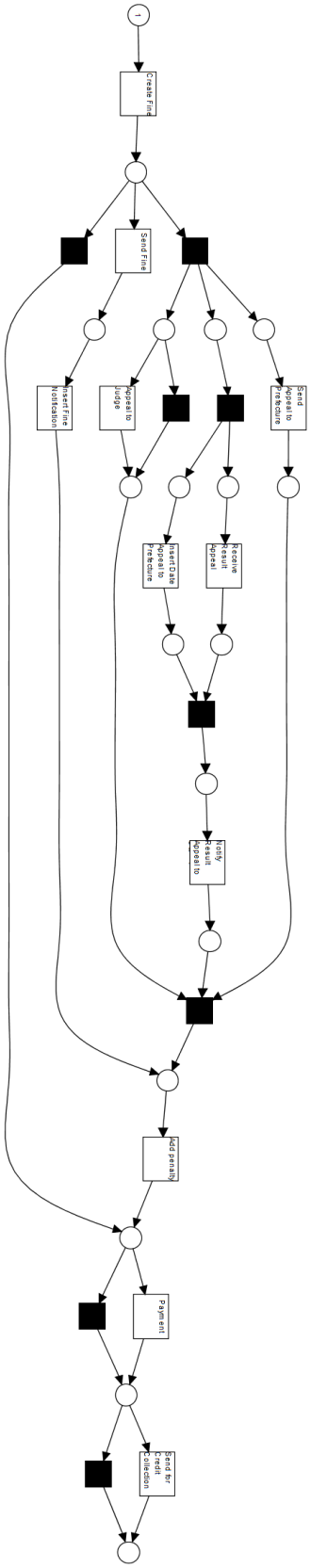
izhodnem pogoju en žeton. Ta žeton lahko sproži tri različne akcije, vendar pa lahko sproži samo eno izmed teh. Prvo lahko sproži skrito akcijo katera potem prenese žeton na konec mreže k možnosti izbire »Payment«, katera potem prenese žeton na možnost izbire še za »Send for Credit Collection«. Drugi možen scenarij je, da se žeton prenese pri aktiviranju akcije »Send Fine« ta se pa potem nadaljuje z »Insert Fine Notification« nato se doda še dodatno plačilo »Add Penalty« ter potem spet zaključi enako kot pri prvem primeru. Nato pa sledi še tretja možnost. Kjer se aktivira skrita akcija, ki ustvari na vseh svojih izhodih po en žeton. Se pravi, da se vzporedno začnejo izvajati trije različni postopki. Prvi izmed teh je »Send Appeal to prefecture«, ki prenese žeton na izhodni pogoj. Za nadaljevanje pa se morata zaključiti še ostala dva dela tega postopka. Najprej se žeton prenese na skrito akcijo, katera sproži izvajanje dveh akcij vzporedno in sicer »Receive Result Appeal« in »Insert Date Appeal to Prefecture«. Po zaključku obeh dejanj se izvede še akcija »Notify Result Appeal to Offender«. Za dokončanje tega postopka pa se izvaja še tretji del tega dela procesa, in sicer se lahko izvede akcija »Appeal to Judge«. Ni pa nujno, da se ta izvede, saj kot vidimo na sliki, se lahko sproži skrita akcija in se temu delu izognemo. Komaj takrat ko se vsi ti trije deli izvedejo se lahko nadaljuje z akcijo »Add penalty« in na koncu še z akcija »Payment« in »Send for Credit Collection«, če so izpolnjeni ustrezni pogoji.

9.4.3 BPMN model

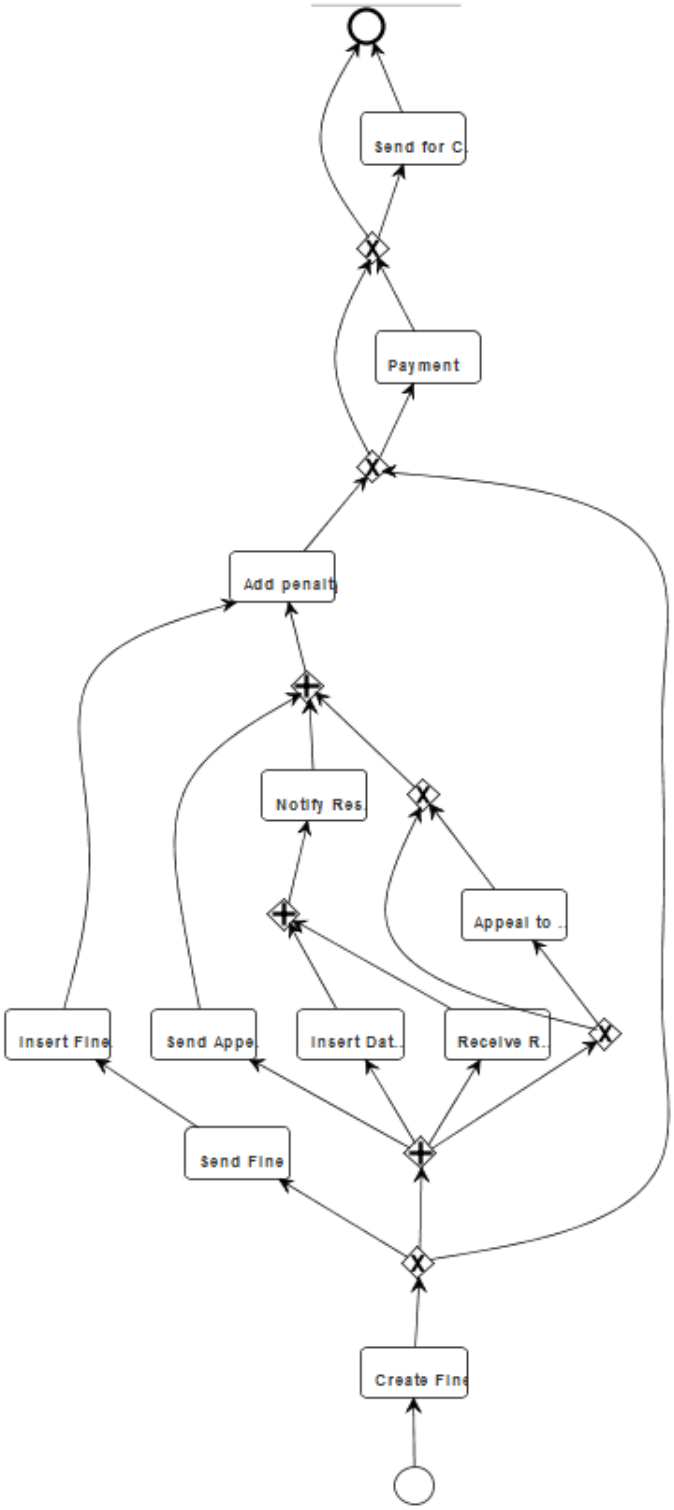
Zelo pogosto se proces predstavi z BPMN modelom. Zaradi tega sem tudi sam pridobljeni model pretvoril v BPMN model. Dobil sem rezultat, kot ga prikazuje slika 9.13.

9.5 Preverjanje skladnosti

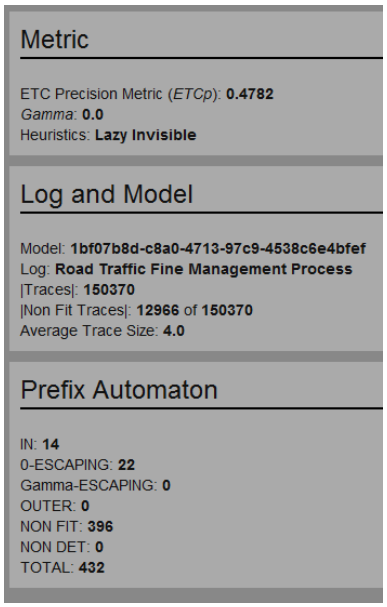
Sedaj, ko sem dobil petrijevo mrežo z inductive minerjem, lahko preverimo na pridobljenih podatkih, katere sledi se ne ujemajo s pridobljeno mrežo. Prav



Slika 9.12: Petrijeva mreža

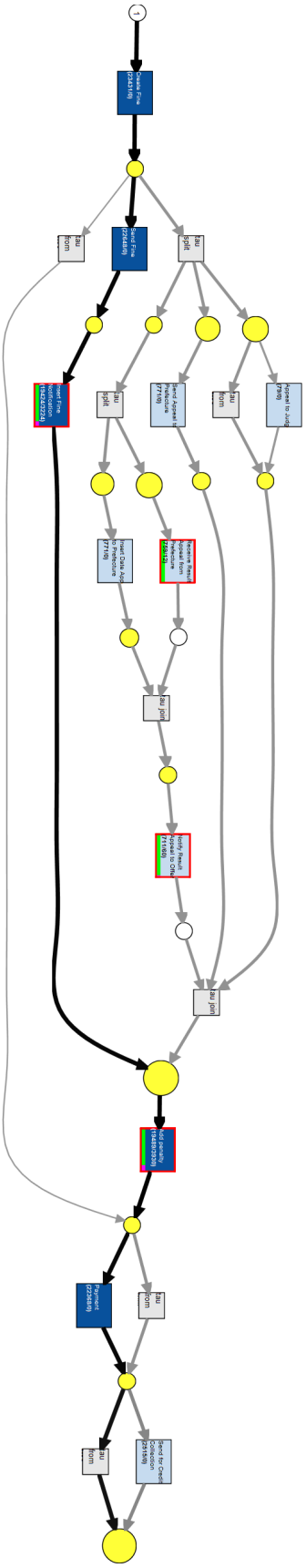


Slika 9.13: BPMN model



Slika 9.14: Natančnost pridobljenega modela

tako pa lahko pregledamo, koliko sledi se ujema ter kakšna je natančnost modela. Z vstavljenim vtičnikov za preverjanje natančnosti dobimo rezultat, ki ga prikazuje slika 9.14. Kot vidimo na izpisu imamo 12966 neprilagojočih primerov od 150370. Se pravi je to natančnost 0,91377 oz. 91,377% primerov smo uspešno rekonstruirali z našim modelom. Kar je precej natančno, celo boljše kot s hevrističnim modelom. Naslednji korak pri preverjanju skladnosti je simuliranje vse primerov skozi model in pogled v primere, v katerih se model razlikuje in zakaj. To sem naredil z vgrajenim vtičnikom ter dobil rezultat, kot ga prikazuje slika 9.15. Na tej sliki so tiste akcije, ki so najbolj obiskane označene s temno modro barvo, redkeje pa s svetlejšo modro, odvisno od intenzitete. Prav tako je z intenziteto odebelitve puščice označena najpogostejša pot v modelu. Pri vsaki aktivnosti, kjer sled ni sledila v skladu z modelom, pa je označeno v kvadratku akcije, z zeleno koliko sledi je uspešno sledilo modelu in z roza barvo koliko jih je bilo neuspešnih. Z rumeno pa so označeni tudi pogoji, kjer je prišlo do premika žetona v katerem od primerov. Prav tako pa imamo kot vidimo na sliki 9.16 možnost pogleda



Slika 9.15: Pregled modela skozi simulacijo

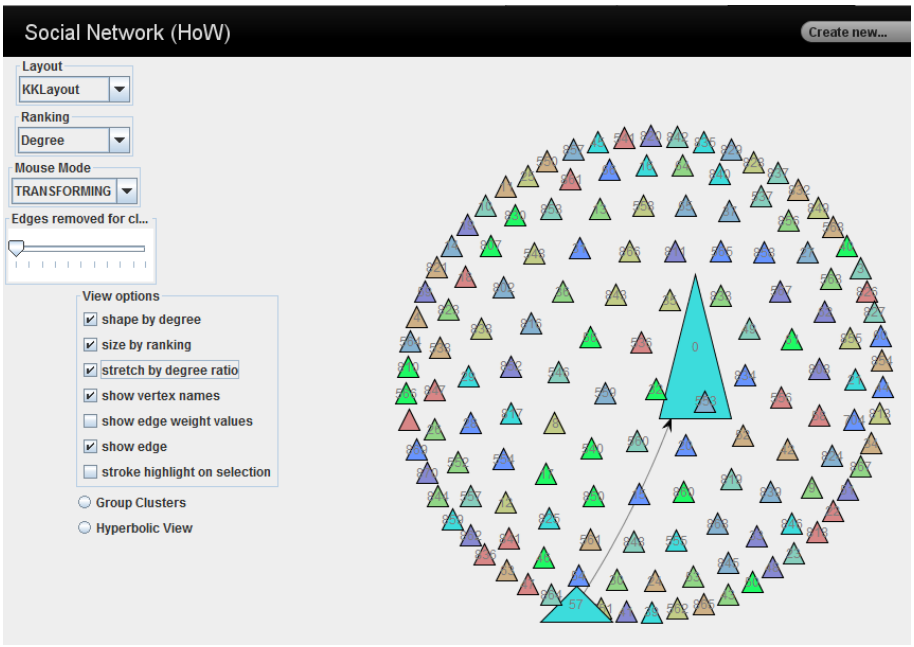
Slika 9.16: Pregled odstopanj sledi

vseh sledi, katera niso ustrezala modelu in jih lahko podrobneje proučimo. Npr. za sled s številko 100395 vidimo, da je zaporedje dogodkov takšno: »Create Fine«, »Send Fine«, »Insert Fine Notification«, »Payment«, »Add penalty«, »Send for Credit Collection«. Se pravi so v tem primeru kršitelju poslali kazen, je ni plačal, v določenem roku so mu poslali opomin, medtem je plačal kazen, vendar pa ni plačal stroškov opomina, zato so mu na koncu poslali še izterjavo za preostanek. Za sled 100385 vidimo, da je dvakrat prišlo do plačila itd. Glede na to, ali so takšna dejanja dejansko lahko dovoljena, popravimo model ali pa sam proces prilagodimo, da do tega ne bo prihajalo.

9.6 Ostale perspektive procesa

9.6.1 Socialna mreža

Kot sem že omenil pri točkastemu diagramu, imamo samo za »Create Fine« podatke o izvršiteljih in kot vidimo na pridobljeni socialni mreži na naših podat-

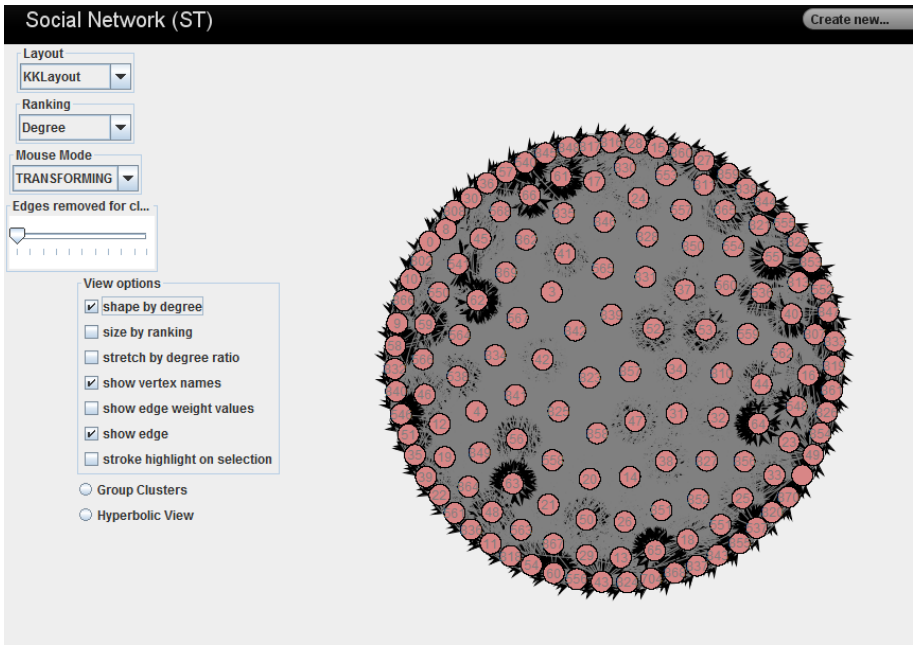


Slika 9.17: Socialna mreža

kih na sliki 9.17, ni nobenega sodelovanja razen med izvrševalcem z številko 0 in 57. Na sliki 9.18 pa vidimo podobne skupine s podobnim opravljanjem dela. Kot vidimo, so vse vključene v enako skupino, saj kot sem omenil, manjka za ostale aktivnosti podatki o izvršiteljih.

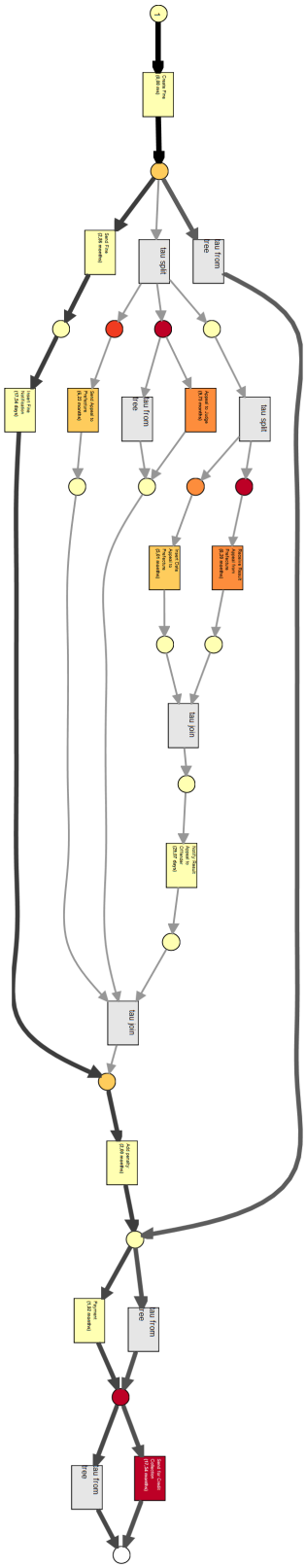
9.6.2 Časovni vidik

Namen časovnega vidika je odkriti ozka grla v sistemu, analizirati časovne parametre ter v glavnem poizkušati izboljšati čas izvajanja procesa. Obstaja več vrst časovnih analiz v programu ProM. Jaz sem uporabil vtičnik »Replay a log on Petri net on performace analysis«. Pri tem sem dobil rezultat, ki ga prikazuje slika 9.19. Kot vidimo na sliki, je najdlje trajajoča aktivnost »Send for Credit Collection«. Povprečni čas trajanja od začetka do zaključka aktivnosti je 17,54 meseca. Prav tako vidimo povprečni čas trajanja aktivnosti za vse ostale aktivnosti. Bolj je barva rdeča, dlje aktivnost traja. Kot vidimo, se aktivnost najhitreje zaključi v primeru takojšnjega plačila, v vseh osta-

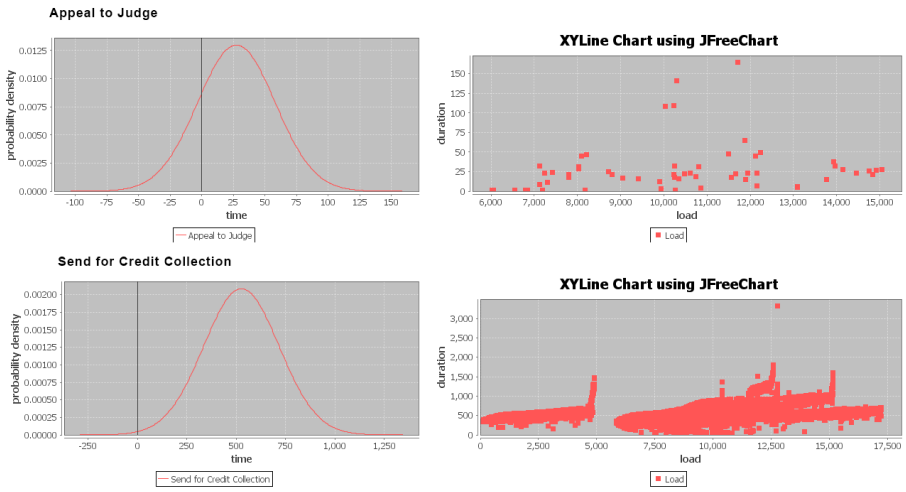


Slika 9.18: Socialna mreža - skupine glede na podobno delo

lih primerih pa zavzame sam proces kar precej časa med prehodom iz ene v drugo aktivnost. V primeru pritožbe na kazen pa se ta čas zelo poveča. Prav tako v primeru zaključka procesa z aktivnostjo »Send for Credit Collection«. Prav tako je v grafu z debelino puščic označeno, katere povezave so najbolje obremenjene. Tukaj vidimo, da je najpogostejši primer takojšnjega plačila kazni ali pa plačilo po prejetju obvestila o potrebnem plačilu. S pregledom grafa lahko takoj vidimo kje so ozka grla ter katera aktivnost traja najdlje. Obstajajo pa še drugi vtičniki za časovno analizo. Eden izmed njih je »Enrich petri net with performace data« s katerim lahko dobimo še malo podrobnejši pregled nad aktivnostjo. Kot vidimo, dobimo takšne rezultate kot je na sliki 9.20 za vsako aktivnost. Prikazano imam pa samo za dve najbolj časovno problematični aktivnosti, to sta »Send for Credit Collection« in »Appeal to Judge«. Na prvem grafu na sliki vidimo, kako je skozi časovno izvajanje v dneh porazdeljena verjetnost pojavitve aktivnosti. Se pravi, kakšna je verjetnost, da bo izid trajanja aktivnosti enak določenemu času. Na drugem



Slika 9.19: Performančne lastnosti petrijeve mreže



Slika 9.20: Podrobnejši prikaz lastnosti dveh aktivnosti

grafu pa vidimo trajanje aktivnosti glede na obremenitev. V našem primeru lahko razberemo, da je za aktivnost »Send for Credit Collection« največja verjetnost, da se zaključi v približno 500 dnevih, saj je takrat vrh pri prvem grafu. Na drugem grafu pa vidimo, da ne glede na količino zahtev, je nekje enakomerno časovno trajanje na primer. Se pravi, je neodvisno od tega ali je polno zahtev ali pa ena zahteva bo približno v enakem časovnem obdobju zaključena. Za aktivnost »Appeal to Judge« pa prav tako pridemo do podobnega zaključka s tem, da se aktivnost konča nekaj časa prej.

9.7 Odgovori na zastavljena vprašanja

1. Koliko je zapisov v dnevniku dogodkov in koliko je dokončanih primerov v njem?

   V pridobljenih podatkih imamo 150370 primerov, ter 561470 dogodkov, vendar pa za analizo rabimo samo dokončane primere, kajti nedokončani primeri nam v nasprotnem primeru samo povzročijo šum v podatkih in s tem pokvarijo analizo, ki jo želimo narediti. Dokončanih primerov in s tem primernih podatkov je 126284 primerov in 499508

dogodkov.

2. Kakšne so lastnosti zapisanih podatkov v dnevniku dogodkov?

   V povprečju se izvedejo 4 različni dogodki na primer. Največ pa se je zgodilo 10 dogodkov na primer. Prav tako pa se tudi v povprečju zgodijo 4 dogodki na primer. Največ dogodkov na en primer,ki se jih je zgodilo pa je 20.

3. Ali je proces strukturiran?

   Skoraj zagotovo lahko trdimo, da je proces strukturiran. To nam dokazuje točkovna analiza, kjer vidimo, da si dogodki v primeru sledijo precej enakomerno, prav tako vidimo enakomerno prihajanje zahtev v sistem, ter njihov zaključek. Kot drugi dokaz pa imamo sam procesni model, v katerem vidimo, da je natančnost modela nad 0,8, saj je nekako nenapisano pravilo, da če je možno ponoviti več kot 80% primerov velja, da je proces strukturiran. Ker je sam proces strukturiran, lahko uporabimo večino tehnik procesnega rudarjenja.

4. Koliko časa traja najdlje izvajajoči primer v procesu in koliko je povprečje trajanja enega primera?

   Najdlje trajajoči primer traja 3285 dni. V povprečju pa traja za dokončanje primera 379 dni, vendar pa nam to kaj dosti ne pove, saj je veliko odstopanje od tega na kakšen način se obravnava primer. Se pravi, če se kršitelj pritoži se čas izvajanja procesa precej podaljša.

5. Kako so aktivnosti med seboj povezane?

   Povezave med aktivnostmi, si najlažje ogledamo na hevurističnem modelu, ki je predstavljen na sliki 9.11 . Prav tako pa lahko pogledamo povezave in delovanje procesa na pridobljeni petrijevi mreži na sliki 9.12 .

6. Ali je mogoče ustvariti procesni model za izbrani proces, ter kako natančen je ta model?

Da, procesni model je mogoče narediti. Sam sem naredil več modelov, vendar pa sem za analizo uporabljal procesni model pridobljen na sliki 9.12 .

7. Kako so izvrševalci aktivnosti v procesu med seboj povezani?

   Kot lahko vidimo na sliki 9.18 in na sliki 9.9 ni povezav med njimi, saj je problem pri zapisu dnevnika dogodkov. Saj je samo za začetni proces zabeležen izvršitelj. Zaradi tega ni nobenega predaja dela. Se pravi, da ne moremo na to vprašanje odgovoriti zaradi pomanjkanja podatkov.

8. Katere aktivnosti trajajo najdlje v procesu oz. kje so ozka grla v procesu?

   Najdlje traja proces »Send for Credit Collection«. Vsak primer, kjer se kršitelj pritoži, se sam proces obravnave kazni zelo zavleče. Se pravi, če želimo skrajšati povprečen čas izvajanja, moramo najprej skrajšati čas izvajanja »Appeal to Judge«, večina primerov, ki se obravnavajo s to aktivnostjo se tudi končajo z aktivnostjo »Send for Credit Collection«. To pa za takšne primere zelo poveča čas izvajanja procesa.

9.8 Zaključek

V diplomski nalogi smo pregledali področje procesnega rudarjenja. Prav tako pa smo predstavljene tehnike uporabili na realnih podatkih iz procesa. Kot rezultat pa smo dobili različne predstavitve podatkov, od modelov do grafov, ter mrež. Te rezultate pa lahko uporabimo za analizo procesa ter njegovo izboljšanje.

Ugotovili smo, da je za dobro procesno rudarjenje potrebno poznati pomanjkljivosti in prednosti posameznega algoritma. Prav tako pa je potrebno pravilno interpretirati podatke. Na koncu pa je še vseeno velik del analize odvisen od samega izvajalca analize in njegove interpretacije pridobljenih rezultatov. Zato nam procesno rudarjenje ne reši vseh problemov pri izgradnji

modelov, je pa odličen način, da dobimo dodatne informacije v zvezi s tem, kaj se v realni postavitvi procesa dogaja. Poleg vseh ostalih stvari pa je pomembno začetno filtriranje podatkov, saj v nasprotnem primeru dobimo modele, kateri ustrezajo tudi nedokončanim podatkom, oz. podatkom, ki so rezultat napak v beleženju dnevnikov dogodkov(šum). Seveda pa morajo biti podatki tudi v pravilni obliki, saj v nasprotnem primeru ne moremo izvesti procesnega rudarjenja.

Z uporabo programa ProM sem pri analizi procesa lahko odgovoril na večino vprašanj, ki sem si jih na začetku analize zastavil. Zaradi tega menim, da je diplomska naloga dosegla svoj cilj, ki je bil predstaviti prednosti procesnega rudarjenja za analizo procesov. Prav tako pa sem predstavil prosto dostopen program ProM, kateri je lahko še kako uporaben pri izboljšanju procesov. Glavni problem pri uporabi orodja je večinoma pridobiti ustrezno zapisane podatke, saj so ponavadi porazdeljeni v več bazah, če sploh so, zato se v večini novejših informacijskih sistemih začenja uporaba beleženja podatkov na nivoju procesov, ter nato že sam informacijski sistem poskrbi za pravilni zapis podatkov.

Možne izboljšave oz. nadaljevanje diplomske naloge bi lahko bile, pregled analize procesnega rudarjenja medtem, ko se podatki še beležijo. S tem bi lahko dobili oceno še potrebnega časa za izvedbo primera ali pa če pride do kakih kršitev modela in podobno. Naslednja možna izboljšava bi lahko bila uporaba teh sklepov za dejanski poizkus izboljšanja analiziranega procesa.

Pri analizi sem uporabil znanje pridobljeno z več področij študija na Fakulteti za računalništvo in informatiko, katerega sem potem uporabil za uspešno analizo s procesnim rudarjenjem, za katero menim, da bi morala biti vgrajena v večino večjih informacijskih sistemov.

Literatura

[1] D. Stopar, "Upravljanje poslovnih procesov", *diplomsko delo*, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Ljubljana 2009

[2] Wikipedia: "Process mining", Dostopno na: http://en.wikipedia.org/wiki/Process_mining

[3] Will M.P. van der Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes", Springer, Berlin Heidelberg 2011

[4] Supporting the Full BPM Life-Cycle Using Process Mining and Intelligent Redesign, Dostopno na: http://www.processmining.org/_media/publications/wvdaalst_p381.pdf

[5] Process Mining: Making Knowledge Discovery Process Centric, Dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.401.8574&rep=rep1&type=pdf

[6] Projektiranje informacijskih sustava, Modeliranje procesa Dostopno na: http://adria.fesb.hr/ zmiletic/Projektiranje%20informacijskih%20sustava/6.%20Procesni%20model.pdf

[7] Wikipedia: "Process modeling", Dostopno na: http://en.wikipedia.org/wiki/Process_modeling

[8] Petrijeve mreže, Dostopno na: http://lrss.fri.uni-lj.si/sl/teaching/mis/lectures/MIS.pdf

[9] Event logs, Dostopno na:
http://www.processmining.org/logs/start

[10] Extensible Event Stream - Standard Definition, Dostopno na:
http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf

[11] An Introduction to the XES Standard, Dostopno na:
http://fluxicon.com/blog/2010/09/intro-to-xes/

[12] Alpha Algorithm, Dostopno na:
http://0agr.ru/wiki/index.php/Alpha_Algorithm

[13] ProM Tips — Which Mining Algorithm Should You Use?, Dostopno na:
http://fluxicon.com/blog/2010/10/prom-tips-mining-algorithm/

[14] A.E. Eiben in J.E. Smith. "Introduction to Evolutionary Computing. Natural Computing". Springer-Verlag, Berlin, 2003.

[15] Genetic Process Mining, Dostopno na:
http://is.tm.tue.nl/staff/aweijters/p266.pdf

[16] J. Carmona 1 , J. Cortadella 1 in M. Kishinevsky "A Region-based Algorithm for Discovering Petri Nets from Event Logs", Universitat Polit' ecnica de Catalunya, Spain, Intel Corporation, USA

[17] Process mining and fraud detection, Dostopno na:
http://essay.utwente.nl/62633/1/MSc_JJ_Stoop.pdf

[18] Wil M.P. van der Aalst "Process Mining in the Large: A Tutorial". Eindhoven University of Technology, Eindhoven, The Netherlands, 2014.

[19] Towards Comprehensive Support for Organizational Mining, Dostopno na:
http://wwwis.win.tue.nl/ wvdaalst/old/publications/p484.pdf

[20] Business process mining: An industrial application, Dostopno na:
http://www.sciencedirect.com/science/article/pii/S0306437906000305

[21] Supporting Process Mining by Showing Events at a Glance, Dostopno na:
http://www.processmining.org/_media/publications/song2007.pdf

[22] ProM, Dostopno na:
http://www.processmining.org/prom/start

[23] The Prom framework: A new era in process mining tool support, Dostopno na:
http://wwwis.win.tue.nl/ wvdaalst/publications/p264.pdf

[24] ProM 6: The Process Mining Toolkit, Dostopno na:
http://ceur-ws.org/Vol-615/paper13.pdf

[25] Process Mining Tools: A Comparative Analysis, Dostopno na:
http://alexandria.tue.nl/extra1/afstversl/wsk-i/ailenei2011.pdf

[26] Petrijeve mreže, Dostopno na:
http://lrss.fri.uni-lj.si/sl/teaching/omis/lectures/3_Petrijeve_mreze.pdf

[27] Petri nets, Dostopno na:
http://www.emse.fr/ xie/SJTU/Ch3.ppt

[28] Soundness of Workflow Nets: Classification, Decidability, and Analysis, Dostopno na:
http://wwwis.win.tue.nl/ wvdaalst/publications/p628.pdf

[29] Causal Nets: A Modeling Language Tailored Towards Process Discovery, Dostopno na:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.297.4206
&rep=rep1&type=pdf

[30] On the Representational Bias in Process Mining, Dostopno na:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.8701
&rep=rep1&type=pdf

[31] PROCESS MINING, Dostopno na:
https://www.wiso.uni-hamburg.de/fileadmin/wiso
_fs_wi/Publikationen/Michael/Gehrke_und_Werner_-_2013_-
_Process_Mining_Pre-print_Version.pdf

[32] Methods for the specification and verification of business processes, Dostopno na:
http://www.cli.di.unipi.it/ rbruni/MPB-12/24-Mining.pdf

[33] Social Network Analysis for Business Process Discovery, Dostopno na:
https://fenix.tecnico.ulisboa.pt/downloadFile/395142123919/Claudia-Alves-55815-Disserta%C3%A7%C3%A3oFinal.pdf

[34] ProM, Dostopno na:
http://www.processmining.org/prom/start

[35] Road Traffic Fine Management Process, Dostopno na:
http://data.3tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[36] Discovering Block-Structured Process Models From Event Logs - A Constructive Approach, Dostopno na:
http://bpmcenter.org/wp-content/uploads/reports/2013/BPM-13-06.pdf

[37] Discovering Block-Structured Process Models from Incomplete Event Logs, Dostopno na:
http://bpmcenter.org/wp-content/uploads/reports/2014/BPM-14-05.pdf